IntelliJ IDEA 2016.2 Help

Meet IntelliJ IDEA

Welcome to IntelliJ IDEA help!

JetBrains IntelliJ IDEA is cross-platform. It works on Windows, OS X and Linux and brings the whole range of precise developers tools, all tied together to create the convenient development environment.

- If you are completely new to IntelliJ IDEA, read Discover IntelliJ IDEA . The links in this section will lead you to more detailed instructions.
- Before starting with IntelliJ IDEA, take a look at the essentials , since they are useful for more productive usage of IntelliJ IDEA.
- If you want to learn about the IntelliJ IDEA UI, then take a guided tour . The parts Tool Windows and IntelliJ IDEA Editor tell you everything you need to know to get a grip of these UI elements.
- If you click Help button on a dialog, you will see the reference page that gives descriptions of controls and fields of each dialog.
- Besides the dialog descriptions, the part Reference also keeps miscellaneous important information, like icons reference , index of menu items , version control reference , etc.
- The tutorials for the newbies are in the part Java SE . They help perform the most important basic tasks with IntelliJ IDEA - create and run an application.
- For those who want to learn more about IntelliJ IDEA, the part General Guidelines provides information about creating projects, configuring their structure, output etc.
- The more advanced users will find it useful to read the various sections of the part Language and Framework - Specific Guidelines .
- The part Migration Guides is intended for Eclipse and NetBeans users, who are about to move to IntelliJ IDEA.
- Finally, the part Getting Help tells about using help topics in the online and built-in variants, applying to support service, reporting issues and sharing your feedback.

Requirements for IntelliJ IDEA

In this section:

- Hardware Requirements
- System Requirements
  - Windows
  - OS X
  - Linux

Tip To get acquainted with IntelliJ IDEA, you can download and install its trial version for free. This trial version is available for 30 days, whereupon you need to obtain and register a license.

Hardware Requirements

- 1 GB RAM minimum, 2 GB RAM recommended
- 300 MB hard disk space + at least 1 GB for caches
- 1024x768 minimum screen resolution

System Requirements

JRE 1.8 is bundled with distributions for all platforms. You don't need any Java to be installed on your machine in order to run IntelliJ IDEA.

JDK is not required for developing Web, PHP, Ruby or Python projects in IntelliJ IDEA. The standalone JDK installation is required for Java development only.

However, you CANNOT use our bundled JRE for Java development.

Windows

System Requirements

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- To develop Java applications with IntelliJ IDEA, install JDK version according to your project requirements.
  Note that you can't use JRE bundled with IntelliJ IDEA for development, only to run the IDE itself.

Installation

1. Download IntelliJ IDEA from the Download page.
2. Run the `ideaIC or ideaIU-*.exe` file that starts the Installation Wizard.
3. Follow all steps suggested by the wizard. Pay special attention to the corresponding installation options.

OS X

System Requirements

- OS X 10.5 or higher, including 10.10
- Only 64-bit OS X is supported.
- To develop Java applications with IntelliJ IDEA, install JDK version according to your project requirements.
  Note that you can't use JRE bundled with IntelliJ IDEA for development, only to run the IDE itself.

Installation

1. Download the `ideaIC or ideaIU-*.dmg` OS X Disk Image file from the Download page.
2. Double-click the downloaded `ideaIC or ideaIU-*.dmg` OS X Disk Image file to mount it.
3. Copy IntelliJ IDEA to your Applications folder.

Note: Open JDK 1.8 customized by the JetBrains team is bundled in OS X distribution.

Linux

System Requirements

- OS Linux 64 bit
- KDE, GNOME or Unity DE desktop
- To develop Java applications with IntelliJ IDEA, install JDK version according to your project requirements.
  Note that you can't use JRE bundled with IntelliJ IDEA for development, only to run the IDE itself.

Installation

1. Download the `ideaIC or ideaIU-*.tar.gz` file from the Download page.
2. Unpack the `ideaIC or ideaIU-*.tar.gz` file to an empty directory using the following command:

```
tar xfz ideaIC or ideaIU-*.tar.gz
```

3. Because running IntelliJ IDEA from wherever you downloaded the file to may be inconvenient, it is recommended that you move the extracted or unpacked archive folder to the desired location using the `mv` command in one of the following formats:
   -
```
mv <path to extracted
                archive
                folder> <new archive
                folder>
```

     For example:

```
mv /downloads/ideaIC or ideaIU-* my/desired/location
```

   -
```
mv <path to ideaIC
                or
                ideaIU-*.tar.gz> <new archive
                folder>
```

     For example:

```
mv /downloads/ideaIC or ideaIU-*.tar.gz my/desired/location
```

4. Switch to the `bin` directory in the new location:

```
cd <new archive folder>/ideaIC
                or ideaIU-*/bin
```

For example,

```
cd my/desired/location/ideaIC or ideaIU-*/bin
```

5. Run `ideaIC or ideaIU.sh` from the `bin` subdirectory.

Note: Open JDK 1.8 customized by JetBrains is bundled in Linux distribution. In case you are not happy with it, feel free to switch to another JRE installed in your system.

Installing and Launching

On this page:

- IntelliJ IDEA editions
- Downloading and installing IntelliJ IDEA
- Starting IntelliJ IDEA on the various platforms
  - Starting IntelliJ IDEA on Windows
  - Starting IntelliJ IDEA on OS X
  - Starting IntelliJ IDEA on Linux
    - Additional launchers
      - Creating a desktop shortcut
- Importing IntelliJ IDEA usage preferences from a previous version
- For Eclipse and NetBeans users

Tip To get acquainted with IntelliJ IDEA you can download and install its trial version for free. This trial version is available for 30 days, whereupon you need to obtain and register a license.

IntelliJ IDEA editions

There are two IntelliJ IDEA editions: Community and Ultimate.

The Community Edition is free and open source but has less features. The Ultimate Edition is commercial but provides an outstanding set of tools and features. (You can evaluate IntelliJ IDEA Ultimate for free for 30 days.)

For more information, see feature-by-feature Editions comparison matrix.

Downloading and installing IntelliJ IDEA

You can download the latest version of IntelliJ IDEA from the JetBrains website. (There are the versions for Windows, OS X and Linux.)

Earlier versions are available on the Previous IntelliJ IDEA Releases page.

Depending on your operating system:

- Windows: Run the `.exe` file and follow the instructions of IntelliJ IDEA Setup wizard.
- OS X: Open the `.dmg` package, and drag IntelliJ IDEA to the Applications folder.
- Linux: Unpack the `.tar.gz` archive into any directory within your home directory.

Refer to the section Starting IntelliJ IDEA on the various platforms for details.

Starting IntelliJ IDEA on the various platforms

Since 2016.1 IntelliJ IDEA uses Java 1.8.

Starting IntelliJ IDEA on Windows

If during the installation you selected to create a shortcut on your desktop, double-click that shortcut.

Otherwise, go to the `<IntelliJ IDEA>\bin` folder (e.g. `C:\Program Files (x86)\JetBrains\IntelliJ IDEA 2016.1\bin`) and run `idea.exe` or `idea.bat`.

See also, Importing IntelliJ IDEA usage preferences from a previous version.

Starting IntelliJ IDEA on OS X

See Importing IntelliJ IDEA usage preferences from a previous version.

Also, if you wish to change JDK IntelliJ IDEA is running on, use the action Switch boot JDK. To invoke this action, use Searching Everywhere or Find Action.

Starting IntelliJ IDEA on Linux

To launch IntelliJ IDEA on Linux, follow these steps:

1. Download the `ideaIC or ideaIU-*.tar.gz` file from the Download page.
2. Unpack the `ideaIC or ideaIU-*.tar.gz` file to an empty directory using the following command:

```
tar xfz ideaIC or ideaIU-*.tar.gz
```

3. Because running IntelliJ IDEA from wherever you downloaded the file to may be inconvenient, it is recommended that you move the extracted or unpacked archive folder to the desired location using the `mv` command in one of the following formats:

- ```
  mv <path to extracted
                      archive
                      folder> <new archive
                      folder>
  ```

  For example:

  ```
  mv /downloads/ideaIC or ideaIU-* my/desired/location
  ```

- ```
  mv <path to ideaIC
                      or
                      ideaIU-*.tar.gz> <new archive
                      folder>
  ```

For example:

```
mv /downloads/ideaIC or ideaIU-*.tar.gz my/desired/location
```

4. Switch to the `bin` directory in the new location:

```
cd <new archive folder>/ideaIC
                    or ideaIU-*/bin
```

For example,

```
cd my/desired/location/ideaIC or ideaIU-*/bin
```

5. Run `ideaIC or ideaIU.sh` from the `bin` subdirectory.

Additional launchers

Additionally, you can create a desktop shortcut and use it to start IntelliJ IDEA:

– Creating a desktop shortcut

See also, Importing IntelliJ IDEA usage preferences from a previous version.

Creating a desktop shortcut

To create a desktop shortcut for IntelliJ IDEA, use the Create Desktop Entry command. This command is available:

– In the Customize IntelliJ IDEA wizard - when you run IntelliJ IDEA for the first time.

– On the Welcome screen : Configure | Create Desktop Entry.

– In the main menu: Tools | Create Desktop Entry.

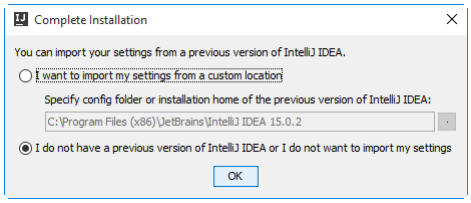
You can choose to create the shortcut:

– For all of your computer users: In this case, the `jetbrains-idea.desktop` entry file is created in `/usr/share/applications` .

– Only for yourself: The `jetbrains-idea.desktop` entry file is created in `~/.gnome/apps` and `~/.local/share/applications` .

Importing IntelliJ IDEA usage preferences from a previous version

When you start IntelliJ IDEA for the first time, the Complete Installation dialog opens, and you have an option of importing IntelliJ IDEA usage preferences and license information from a previous version.



Select one of the following options and click OK.

– I want to import my settings from a previous version (<path>). If this option is present in the dialog, the directory that contains the IntelliJ IDEA settings and license information is found at its default location. (The corresponding path is shown within the option name in parentheses.) Most likely, this is the option that you want.

– I want to import my settings from a custom location. You may have the directory with the settings and license information available but not at its default location. If the option discussed above is missing, IntelliJ IDEA doesn't know where this directory is; the path initially shown in the field is just a guess, and there's no guarantee that the necessary folder is really there. To import the settings in this case, you can specify the folder with the settings (if you know where it is) , or an installation folder of the previous IntelliJ IDEA version. To do that, click … and select the folder in the dialog that opens .

– I do not have a previous version of IntelliJ IDEA or I do not want to import my settings. If this is your case, select this option.

See also, Running IntelliJ IDEA Ultimate for the First Time and Exporting and Importing Settings .

For Eclipse and NetBeans users

Your transition to IntelliJ IDEA may be easier, if you look at these pages:

– Eclipse

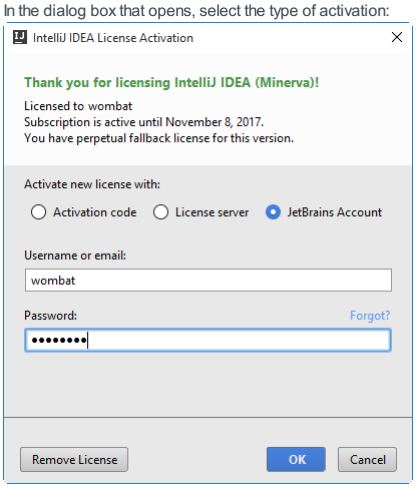
– NetBeans

Registering IntelliJ IDEA

After you've performed installation and launching , you can register IntelliJ IDEA. This is how it's done:

To register IntelliJ IDEA

1. On the main menu, choose Help | Register...
2. In the dialog box that opens, select the type of activation:



   Depending on your selection, do one of the following:

   – Activation code: paste the activation code to the text area.
   – License server: specify the license server address, or click the button Discover server, if the address is not known.
   If you need to learn more, click the link More info... to open the License server documentation .
   – JetBrains Account: select this option if you have the JetBrains Account and wish to register with it.

3. In there are several licenses, choose the one you wish to use:

4. Click OK when ready.

The banner appears in the top-right corner of IntelliJ IDEA that says to whom is the copy of IntelliJ IDEA licensed.

Note that you can delete the unnecessary license. To do that, simpy click the button Remove License in the IntelliJ IDEA License Activation dialog.

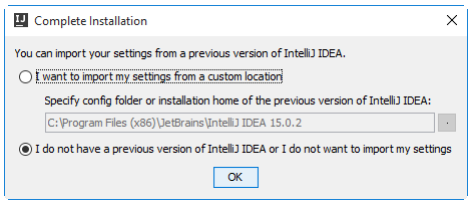
Running IntelliJ IDEA Ultimate for the First Time

Running IntelliJ IDEA Ultimate for the first time assumes performing the following tasks:

- Importing IntelliJ IDEA settings from a previous version
- Selecting to start IntelliJ IDEA evaluation or activating a license
- Accepting the License Agreement
- Selecting the user interface theme
- Disabling unnecessary plugins
- Downloading and installing additional plugins

Importing IntelliJ IDEA settings from a previous version

When you start IntelliJ IDEA for the first time, the Complete Installation dialog opens, and you have an option of importing IntelliJ IDEA usage preferences and license information from a previous version.

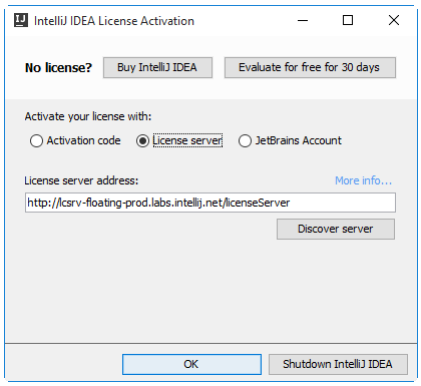


Select one of the following options and click OK.

- I want to import my settings from a previous version (<path>). If this option is present in the dialog, the directory that contains the IntelliJ IDEA settings and license information is found at its default location. (The corresponding path is shown within the option name in parentheses.) Most likely, this is the option that you want.
- I want to import my settings from a custom location. You may have the directory with the settings and license information available but not at its default location. If the option discussed above is missing, IntelliJ IDEA doesn't know where this directory is; the path initially shown in the field is just a guess, and there's no guarantee that the necessary folder is really there. To import the settings in this case, you can specify the folder with the settings (if you know where it is) , or an installation folder of the previous IntelliJ IDEA version. To do that, click and select the folder in the dialog that opens .
- I do not have a previous version of IntelliJ IDEA or I do not want to import my settings. If this is your case, select this option.

The IntelliJ IDEA License Activation dialog opens. (If the license information has been imported from the previous version and is still valid, your most recent project or the Welcome screen opens.)

Selecting to start IntelliJ IDEA evaluation or activating a license



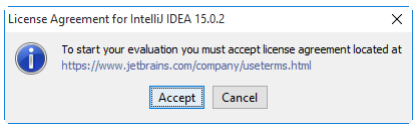
Do one of the following:

- Evaluate for free for 30 days . Click this button to start evaluating IntelliJ IDEA.
- Buy IntelliJ IDEA . Click this button to go to the JetBrains Web site to study the IntelliJ IDEA purchasing options and to buy a license.
- If you are ready to activate your license, do so.

The IntelliJ IDEA License Agreement is shown at the company's site .

Accepting the License Agreement

Read the License Agreement on the site. To accept the license terms, click Accept button.

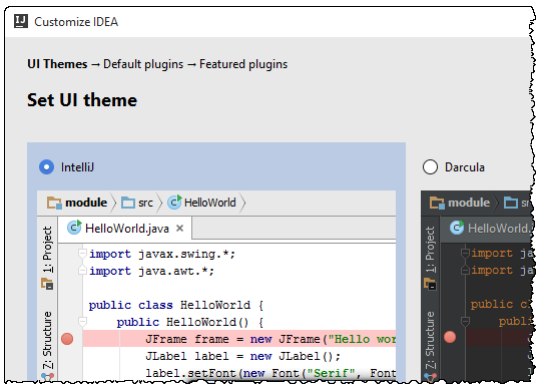


Note that without accepting the license terms, you won't be able to use IntelliJ IDEA.

The Customize IntelliJ IDEA wizard opens.

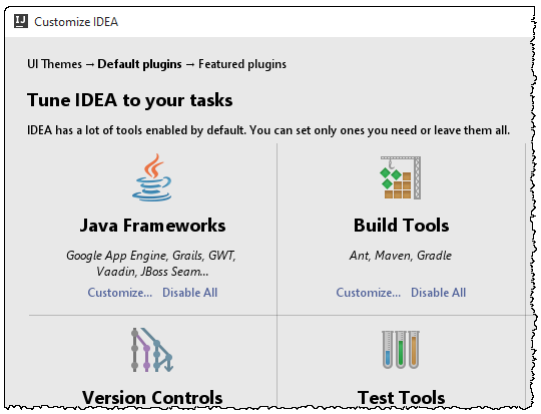
Selecting the user interface theme

Select the user interface theme to be used and click Next.



Disabling unnecessary plugins

On the Default plugins page, you can disable the plugins that you are not going to use. (The more plugins are enabled, the more features you have available. On the other hand, disabling unnecessary plugins may increase the IDE performance, especially on "less powerful" computers.)



The squares on the page correspond to groups of plugins and individual plugins. All these plugins are bundled with the IDE and enabled by default.

Use:

– Customize to disable one or more plugins in a group. (Use the check boxes on the page that opens.)

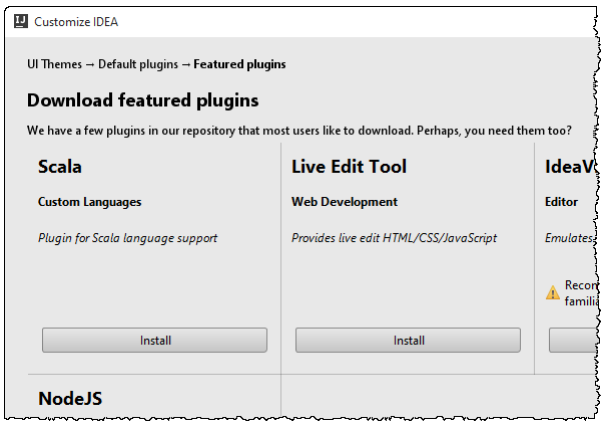
– Disable All or Disable to disable all plugins in a group or an individual plugin.

Note that you can come back to disabling and enabling the plugins at a later time.

Click Next: Featured plugins.

Downloading and installing additional plugins

The Featured plugins page shows the plugins that you might want to download and install. (All these plugins are not included in the IntelliJ IDEA distribution.)



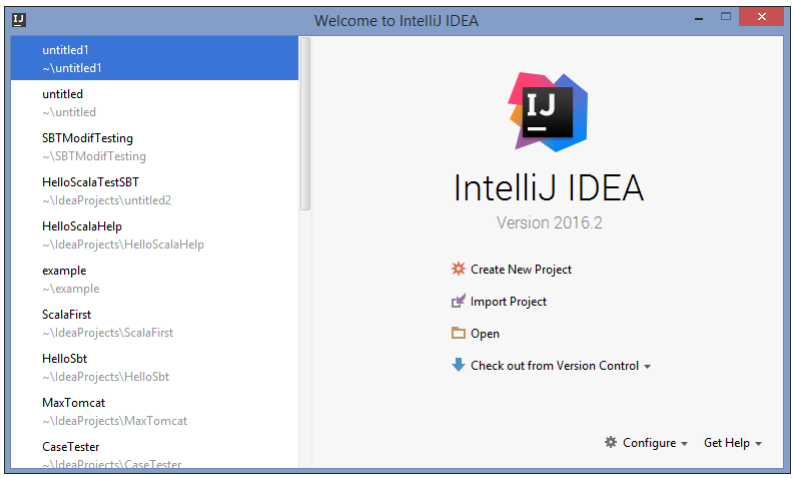
Click Install to download and install anyone of the plugins.

Click Start using IntelliJ IDEA to complete the wizard. As a result, the Welcome screen will open.

Quick Start Guide

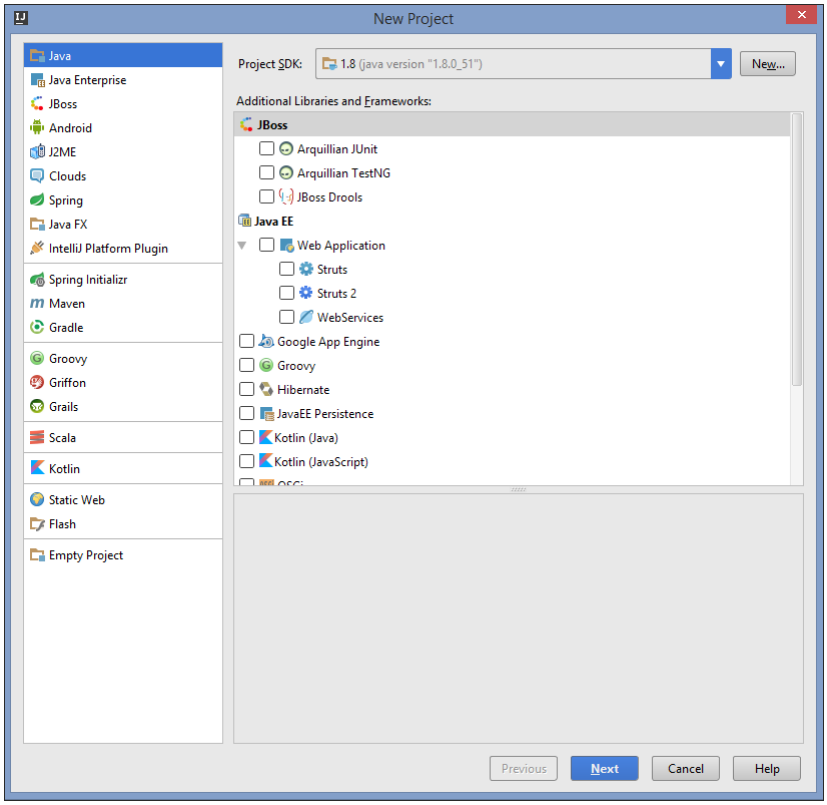
– Run Your First Java Application
– Look Around
– Smart Coding
– Project Configuration in a Nutshell
– Build Your Project
– Run and Debug Your Application
– Run Tests
– Deploy Your Application to an Application Server
– Customize Everything
– Finding Your Way Around
– Migration from Eclipse or NetBeans
– What's Next?

If you have successfully installed Intellij IDEA on your computer, it's time to run it for the first time. You will see the Welcome Screen which gives you main entry points to the IDE. Here you can create a new project, open an existing one, or check out the project from version control.

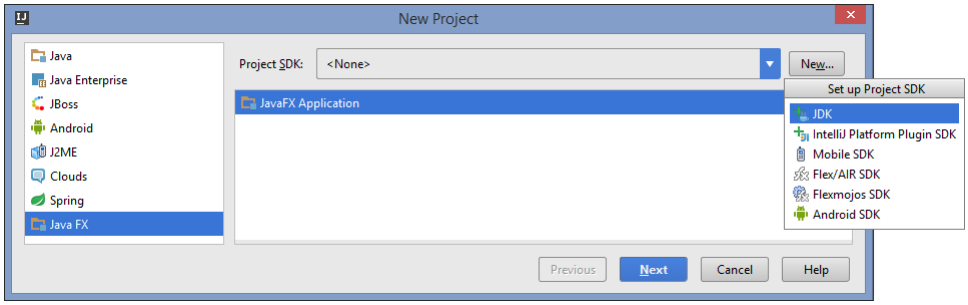


Run Your First Java Application

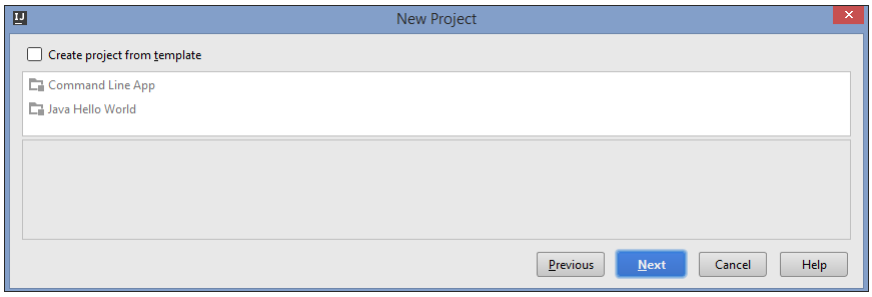
Let's create a simple Java Hello World project. Click Create new project . The New Project Wizard opens.



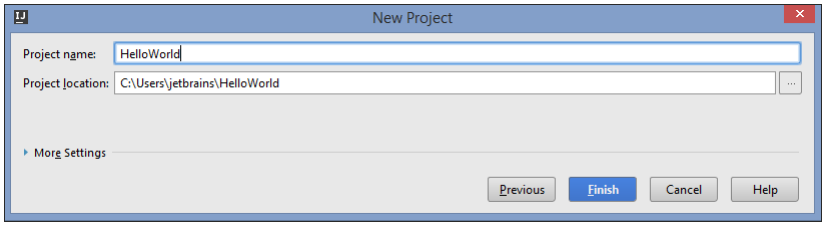
The main thing you should pay attention to is the Project SDK . SDK (Software Development Kit) is a set of software development tools that lets you develop applications. IntelliJ IDEA does not include an SDK, so If you have none, download and install it. Since we make a Java project, we need a Java SDK (JDK). After installation, click New and point to the installation folder of the JDK.



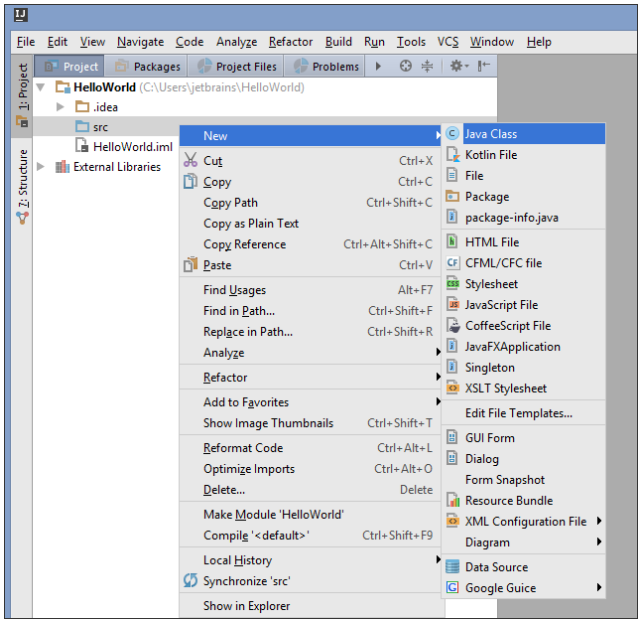
In the New Project Wizard, you can choose technologies your project will support, but as you're making a plain Java Application, select none of them and just click Next.
Then IDE offers you to create project from a template. We don't need this now, so click Next .



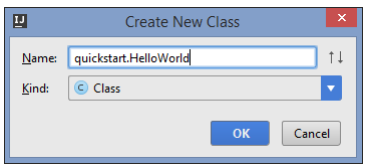
Specify the name and location of the project. Click Finish .



You have src folder inside your project. This is the folder for your source code. Right click on this folder, then choose New | Java Class .



Enter the name of this class, say, quickstart.HelloWorld, and click Finish .

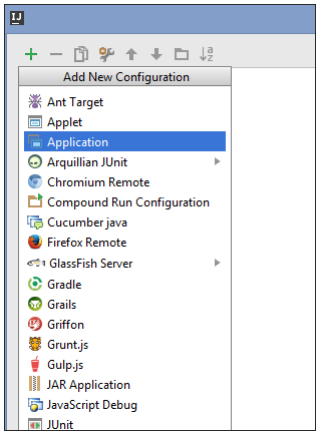


Now the new class opens in the editor. Notice, that HelloWorld class is created in quickstart package. HelloWorld program text is a known one.

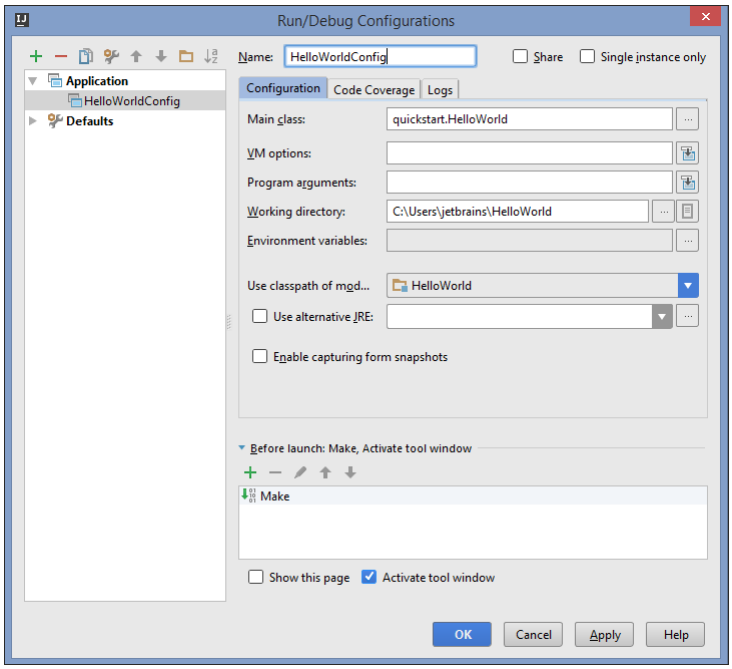
```
package quickstart;

                    public class HelloWorld {
                    public static void main(String[] args)
                    {
                    System.out.println("Hello, World!");
                    }
                    }
```

Put it inside the class.

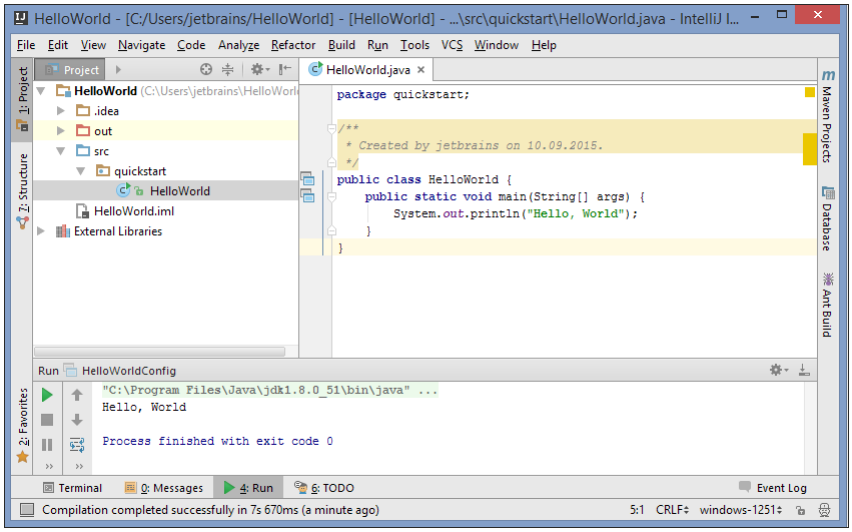
Every time you run an application, you need to have a configuration, which this process is based on. For creating a run configuration click Run | Edit configurations' Then click + and choose Application .



Here you can specify the main class and program arguments. Since we have a simple application, we need to point the Main class only. Put quickstart.HelloWorld into the Main class text field. Print something in Name field, for example, HelloWorldConfig . Run configuration is ready.



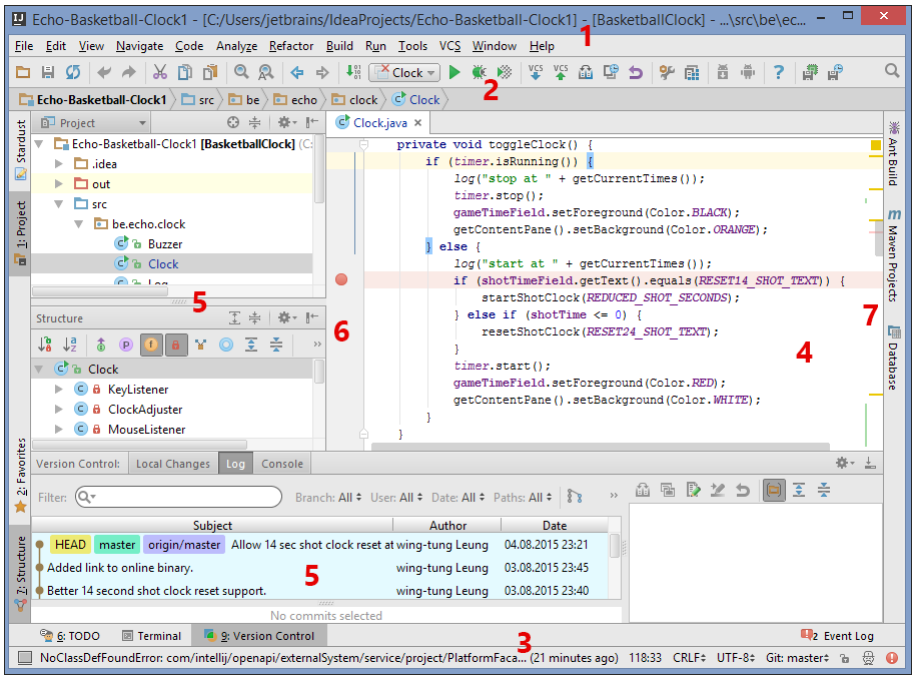
Now you can immediately run your application and make sure all works properly. Choose Run | Run 'HelloWorldConfig' from the main menu. Get a result!



For debugging your application, choose Run | Debug. You should also have Run Configuration for this. To learn more, see Run/Debug Configuration: Application .

Look Around

Let's have a closer look on the main window of IntelliJ IDEA. There are several logical areas:



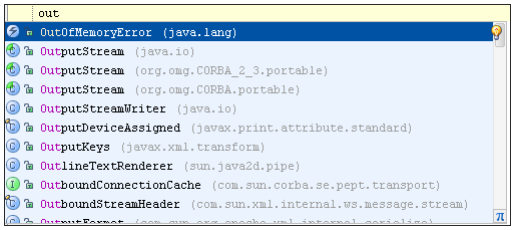
1. Menus and toolbars contain commands that affect the entire project or large portions of it. Additionally, context-sensitive pop-up menus let you perform the commands, which are specific to a part of a project, like source file, class, etc.
2. Navigation bar helps navigate through the project and open files for editing.
3. The status bar indicates the status of your project, the entire IDE, and shows various warning and information messages.
4. The editor where you create and modify the code.
5. Tool windows perform different functions: help you explore and navigate through the projects and file structures, view search and inspection results, run, debug and test applications, work in interactive consoles, and more.
6. Left gutter —the vertical stripe which shows the breakpoints you have, and provides a convenient way to navigate through the code hierarchy like going to definition/declaration. It also shows line numbers and per-line VCS history.
7. Right gutter —this stripe constantly monitors the quality of your code and always shows the results of its code analysis: errors, warnings, etc. The square in the top right-hand corner shows the overall status of code analysis for the whole file.

Smart Coding

IntelliJ IDEA has a powerful editor and always helps you creating error-free applications. There are many smart features in IDE, let's have a look on the most important ones.

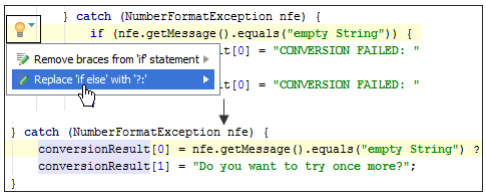
Code completion

Code completion takes into account the current context and saves your time. There are two main types of code completion in IntelliJ IDEA: basic (`Ctrl+Space`) and smart (`Ctrl+Shift+Space`). Basic completion works as you type and completes any name instantly. Smart completion analyzes the context you're currently working in and offers more accurate suggestions based on that analysis. It filters the list of functions and variables to match the expression type.



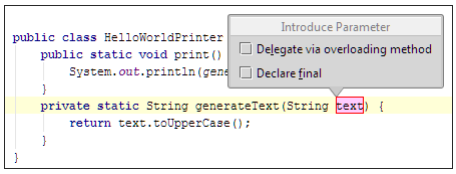
Intention actions

When a possible problem is suspected, IntelliJ IDEA suggests a solution, and in certain cases can implement this solution (properly assign variables, create missing references and more). Besides syntax problems, IntelliJ IDEA recognizes code constructs that can be optimized or improved, and suggests appropriate intention actions, denoted with the special icons. If you want to know exactly what IDE suggests you in the current situation, click the light bulb, or press `Alt+Enter`.



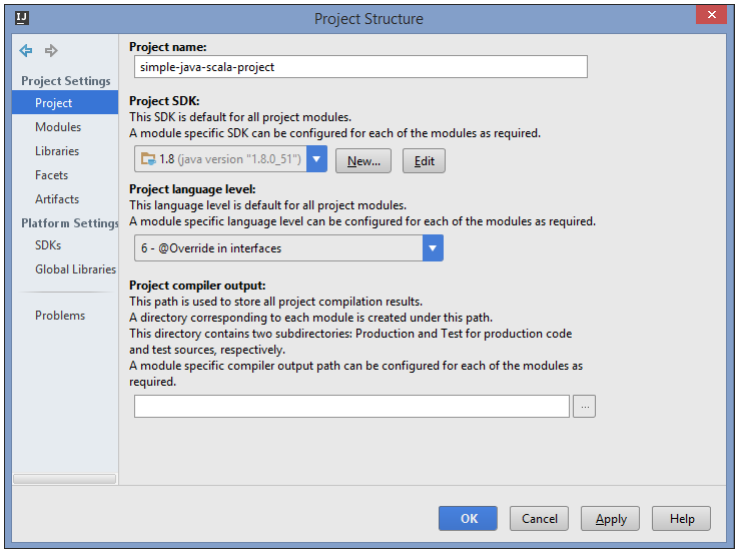
Refactoring

IntelliJ IDEA provides a huge set of automated code refactorings from mere renaming to such complicated things as changing a method signature.



Project Configuration in a Nutshell

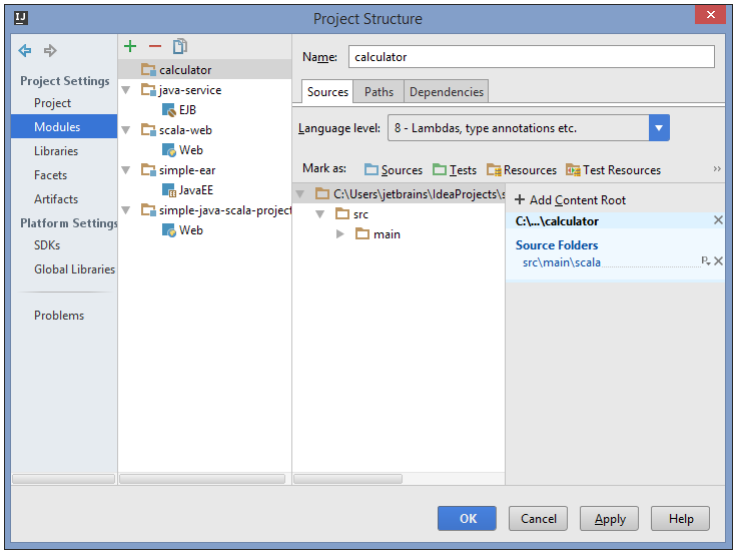
In Intellij IDEA all exists in the context of a Project, the highest level of organization in the IDE. To specify the configuration of your project choose File | Project Structure



Here you can see several items in Project Settings. Let's have a look at them.

Project . Strictly speaking, here are the settings referring to the project generally, where you can configure the name, SDK, language level, and the compiler output path.

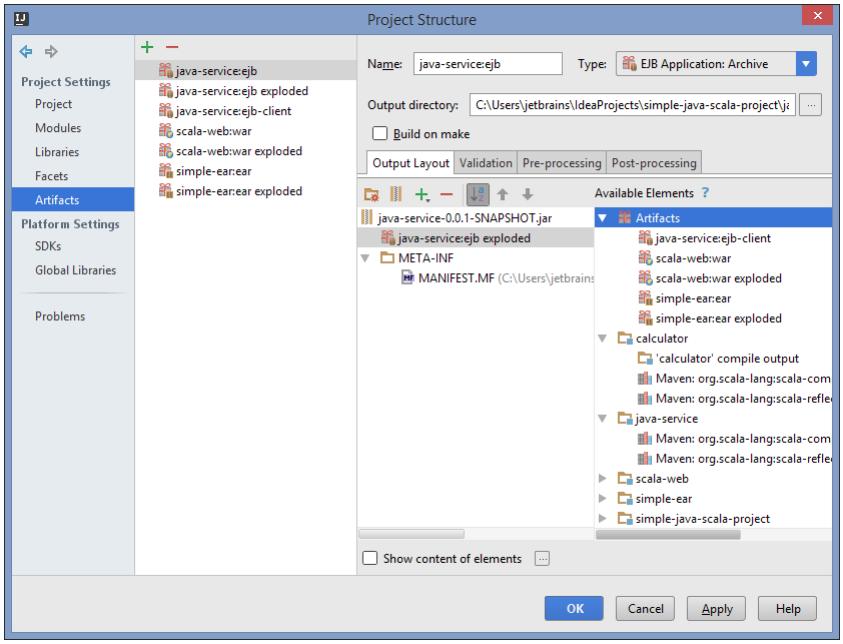
Modules . Every project consists of modules. Module is a discrete unit of functionality that can be compiled, run, debugged and tested independently. Modules contain everything that is required for their specific tasks: source code, build scripts, unit tests, deployment descriptors, and documentation. A module can be a dependency for another module in the project.



Libraries . It is one of the module dependency types in Intellij IDEA, an archive of compiled code that a module can depend on. A Java library, for example, can include class files, archives and directories with class files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`).

Facets . Each module can have a number of facets. Facets are required by the IDE to to provide framework-specific coding assistance, e.g. Web, Spring, Persistence. Mostly, facets are detected automatically by IDE, but here you can configure them manually.

Artifacts . An artifact is the output you want IntelliJ IDEA to generate for your project. It may be a Java archive (`JAR`), Web application archive (`WAR`), an enterprise archive (`EAR`), etc. When you use a build tool, the artifacts are configured automatically, otherwise configure them manually in this section



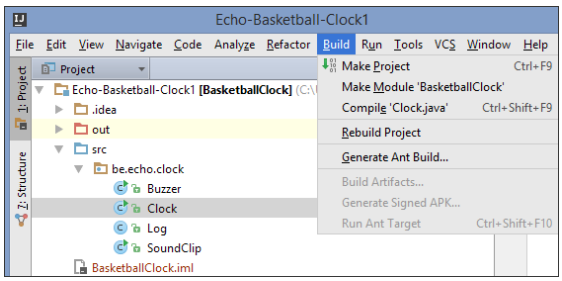
For more information about project structure, read Project Structure Dialog or watch "Project Structure and Configuration" video tutorials.

Build Your Project

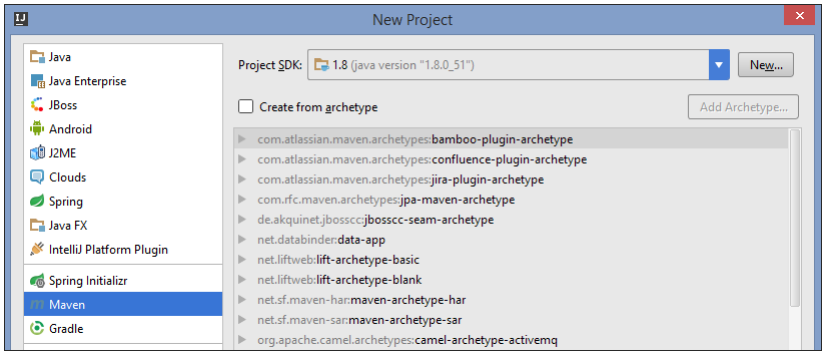
If your project doesn't use a build tool, use Build menu with the following commands:

- Compile: Compiles files in the selected scope, whether they've changed or not.
- Make: Compiles files if they have changed.
- Rebuild Project: Forces a recompile of the entire project.

You can also look at the 'Make, compile and rebuild ' tutorial and see Compilation Types .

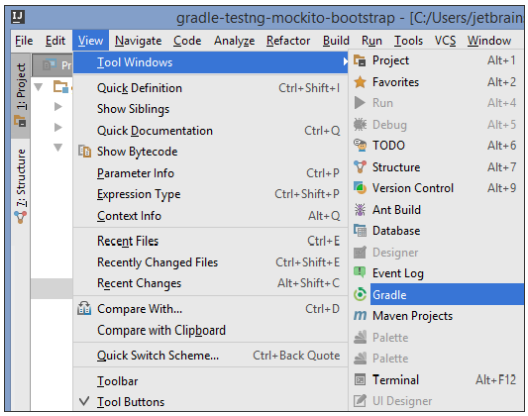


IntelliJ IDEA supports Ant , Maven and Gradle . You can create a new project either by using the Project Wizard and choosing a corresponding build tool, or by importing an existing project from source and choosing the corresponding build file to import from.



If your project uses a build tool, you can also run the tasks of your build tool just from Intellij IDEA. Build menu still works for that kind of projects.

IntelliJ IDEA provides a tool window for a build tool where you can manage specific operations. To open it go to View | Tool Windows | Build Tool Name .



In this window you can also control behavior of the build, run build tasks and add tasks to run configuration as well.

For more information about supported build tools, you can see web help pages about Ant , Gradle or Maven or take a look at "Building on Ant, Gradle and Maven" video tutorial.

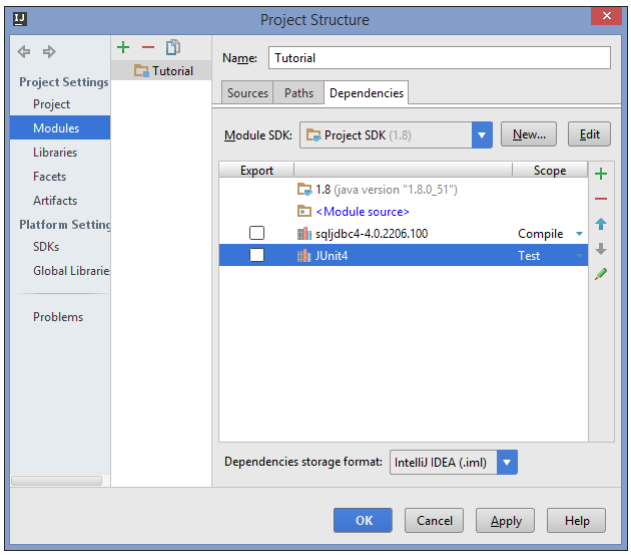
Run and Debug Your Application

Run/Debug configuration describes what happens when you click the Run or Debug button in the Toolbar or from Run menu. You can create several configurations and use the particular one you need by choosing from the dropdown. To edit run configuration click Run | Edit configurations'

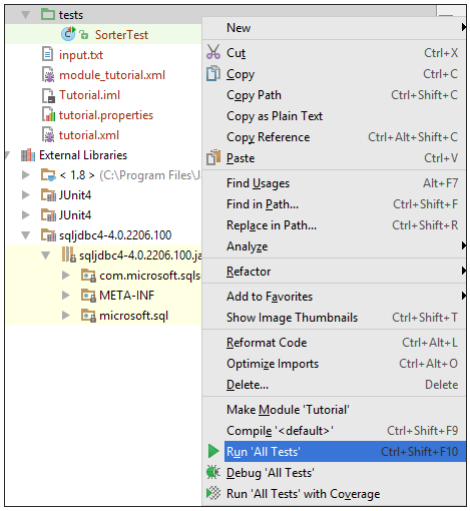
To learn different types of run configurations, please look at the Run/Debug Configurations . See also Run/Debug Configuration basics .

Run and Debug Your Application

Make sure that appropriate test framework is specified in the dependencies of your module.



For running all tests in your application, right-click the test folder and then click Run 'All Tests' .

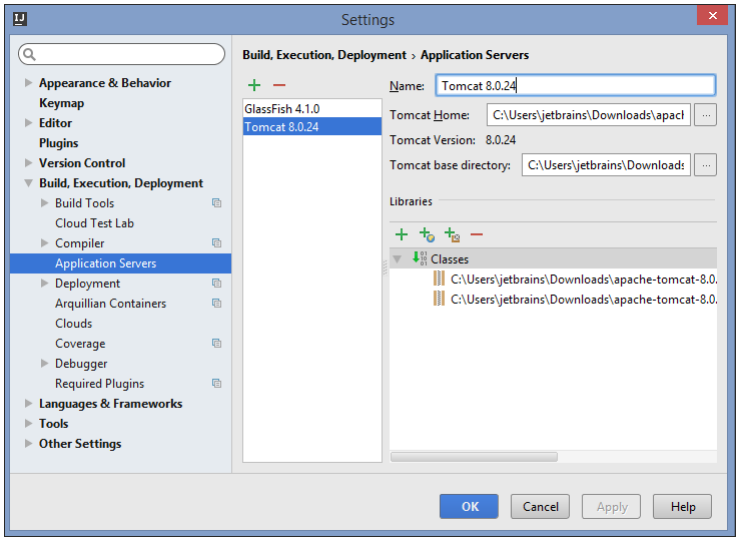


You can run particular test by pressing `Ctrl+Shift+F10` on it. It is also based on run configuration.

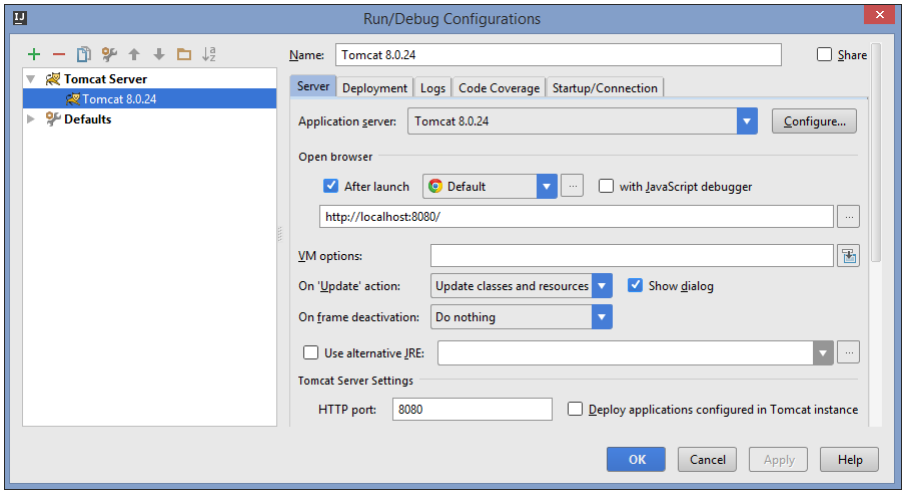
Don't forget to watch Testing video tutorials and read Testing web help page .

Deploy Your Application to an Application Server

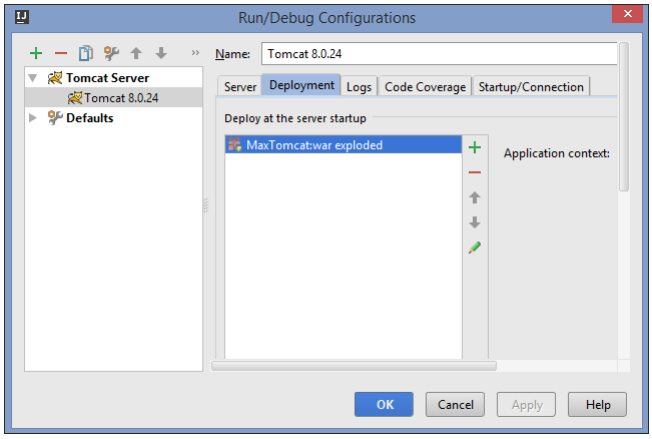
Before you deploy an application to an application server, make sure to configure this application server in File | Settings | Build, Execution, Deployment | Application Servers



Once you have your application server configured, create a corresponding Run/Debug configuration, and specify your application server:



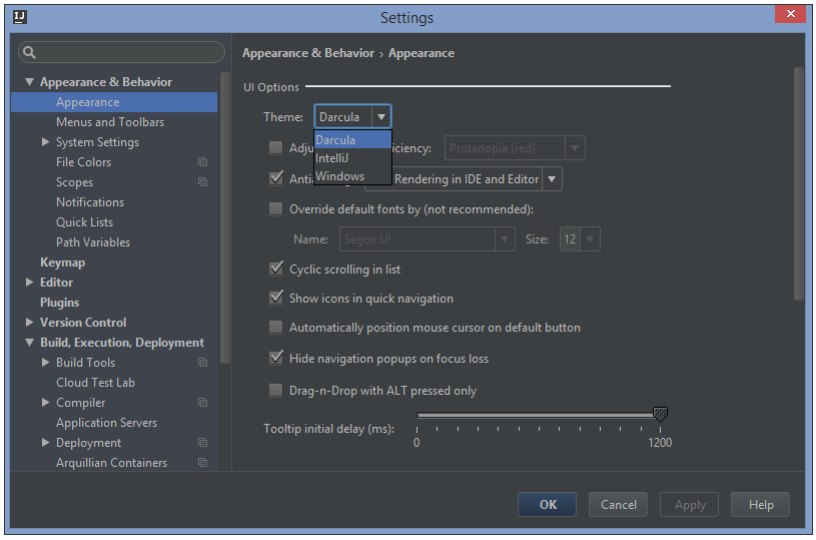
Also, in the Deployment tab specify the actual artifacts to be deployed:



Find more comprehensive tutorial about deploying a Java EE application to an application server at Developing a Java EE Application .

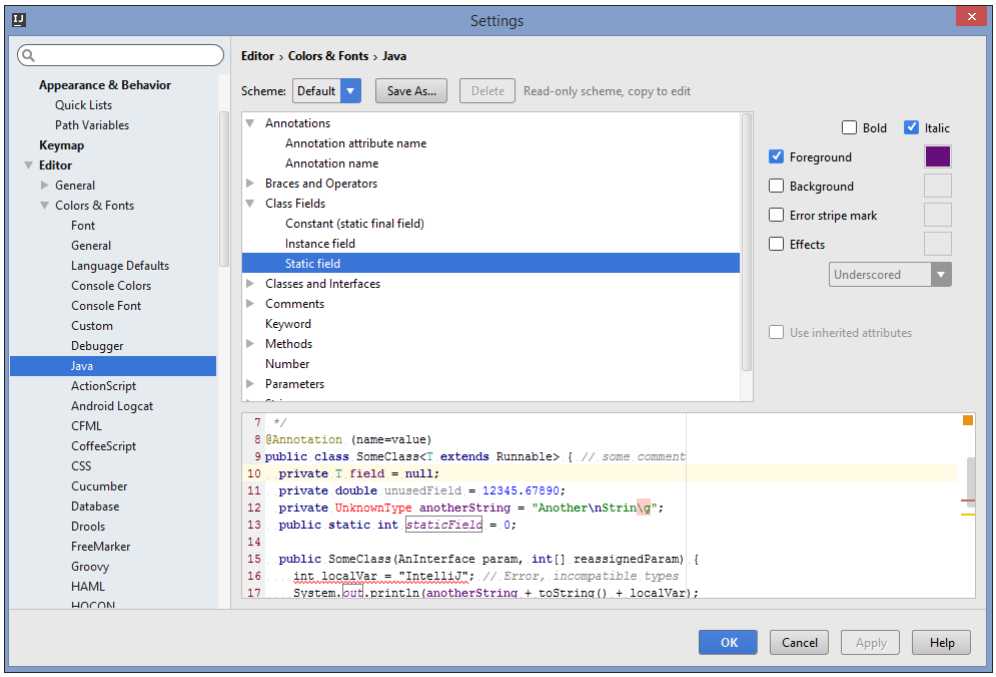
Customize Everything

Customize IDE appearance in File | Settings | Appearance & Behaviour | Appearance. Intellij IDEA provides light and dark appearance themes. There can be several light ones, depending on your OS, and a dark one called Darcula .

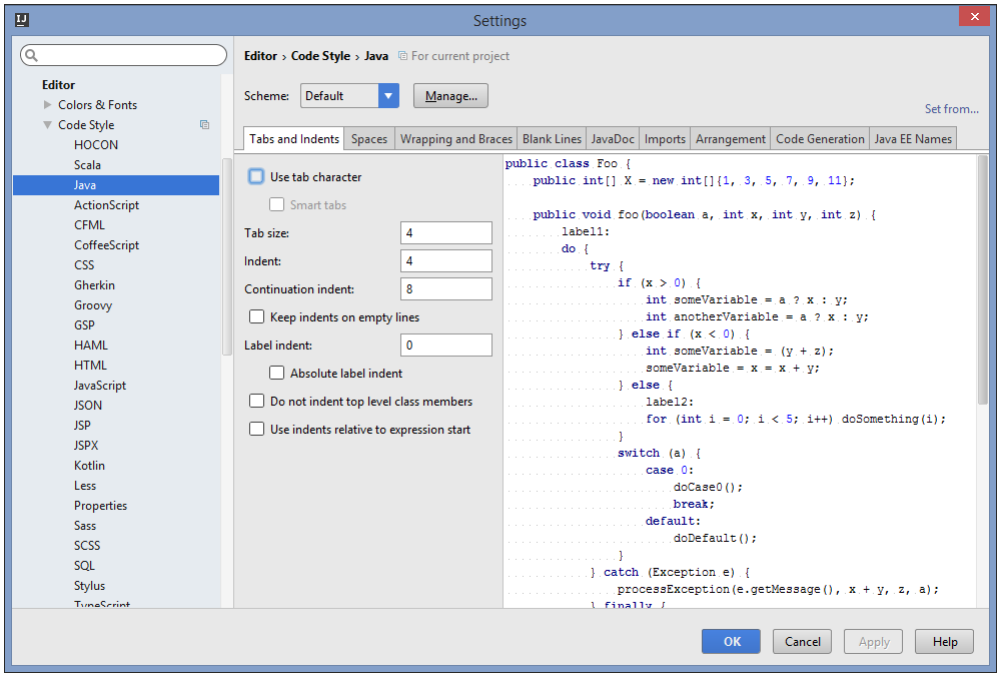


Customize the behaviour of the text editor in File | Settings | Editor .

In File | Settings | Editor | Colors and fonts , you can choose any color scheme for the Editor, modify them and save them as your own.

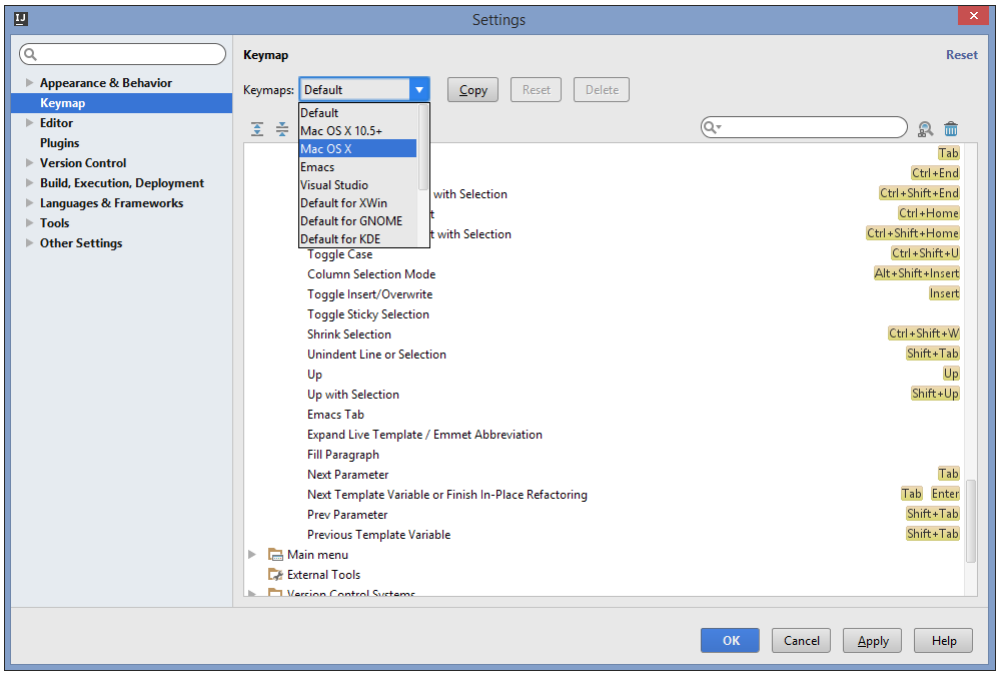


For configuring styles, go to File | Settings | Editor | Code Style . IntelliJ IDEA automatically applies a code style you've configured.



There are lots of shortcuts you can use in the text editor. Begin with checking the shortcuts you cannot miss . Also, download a reference card with most frequently used shortcuts for the various platforms: Win/Linux , MacOS .

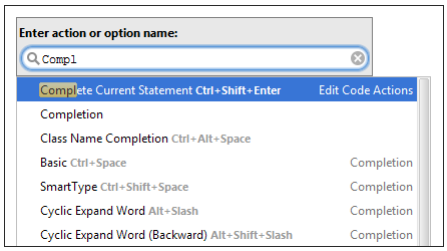
For choosing from several keymaps Intellij IDEA offers you, go to File | Settings | Keymap. You can also assign your own shortcut to any action.



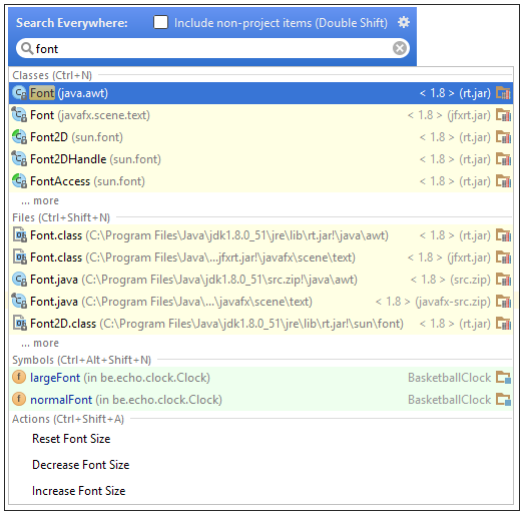
Some customizations can be shared with your team via VCS.

Finding Your Way Around

If you've just forgotten a shortcut, use Find Action (`Ctrl+Shift+A`) and look for action by name.

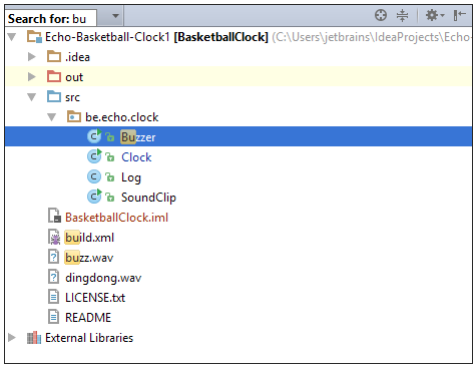


Finally, if you want to find something, but you have no idea where to search, use Search everywhere function by pressing Shift twice.



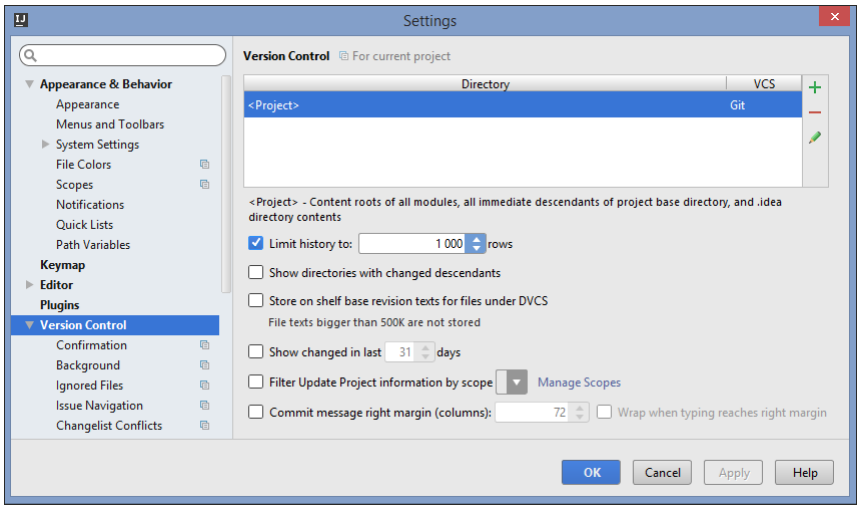
Generally, if you want to find something in settings, use the search bar in the Settings/Preferences dialog .

Try start typing anywhere with trees or lists and find out that speed search is working in many places.

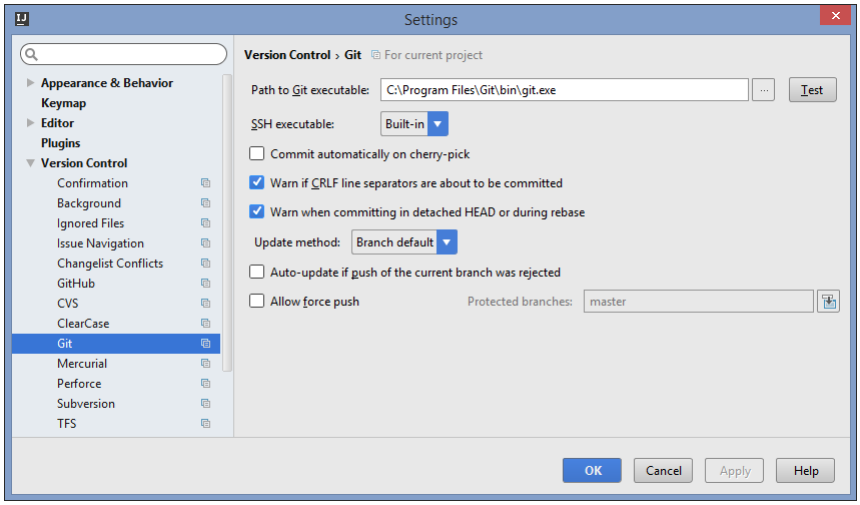


Version Control Integration

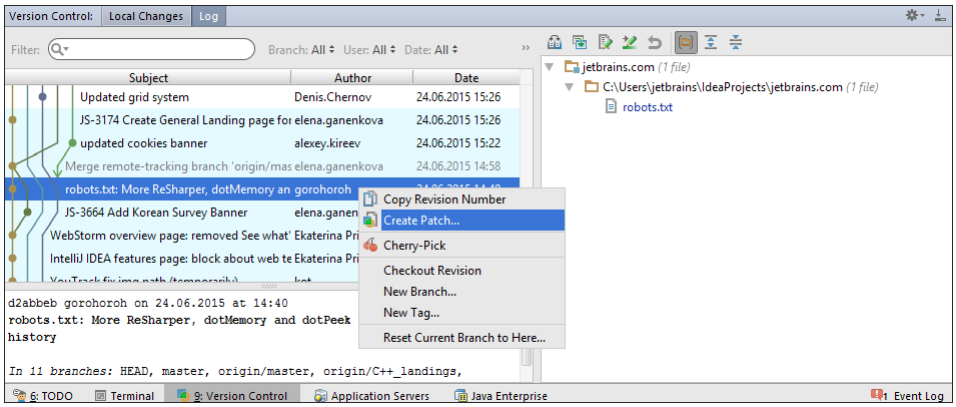
IntelliJ IDEA supports Git, CVS, Mercurial, Perforce and others. Having decided which one to use, specify its settings in File | Settings | Version Control . It's possible to add your current project or an existing project as a new VCS root.



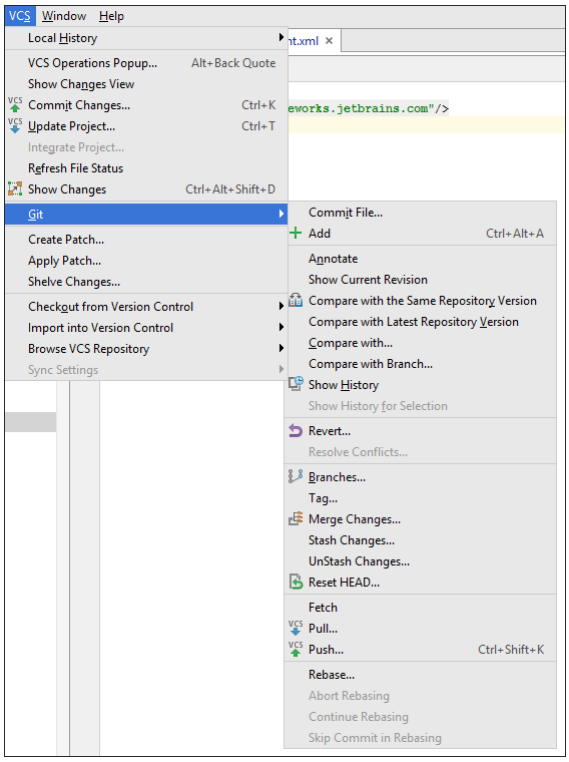
Besides general VCS settings, here you can configure all that corresponds to the particular VCS you are going to use.



IDE lets you add, delete, commit, revert, manage branches, see history, push, pull and do many other things. Some operations are available in Version control tool window called by VCS | Show Changes View .



Others can be called from corresponding point of VCS menu.



To learn more about VCS see Version Control with IntelliJ IDEA and watch Version Control video tutorials .

Migration from Eclipse or NetBeans

We try to do our best to make migration from other IDE's as painless as possible. For those who migrate from Eclipse, there is a detailed migration guide . For those who migrate from NetBeans it is useful to read IntelliJ IDEA for NetBeans Users . Remember that you can keep previous IDE keymap, and also have a look at Importing projects from other IDE video tutorials.

What's Next?

When you complete this guide, have fun watching Tips and tricks video from Hadi Hariri. Feel free to share your feedback in our discussion forum and issue tracker . We

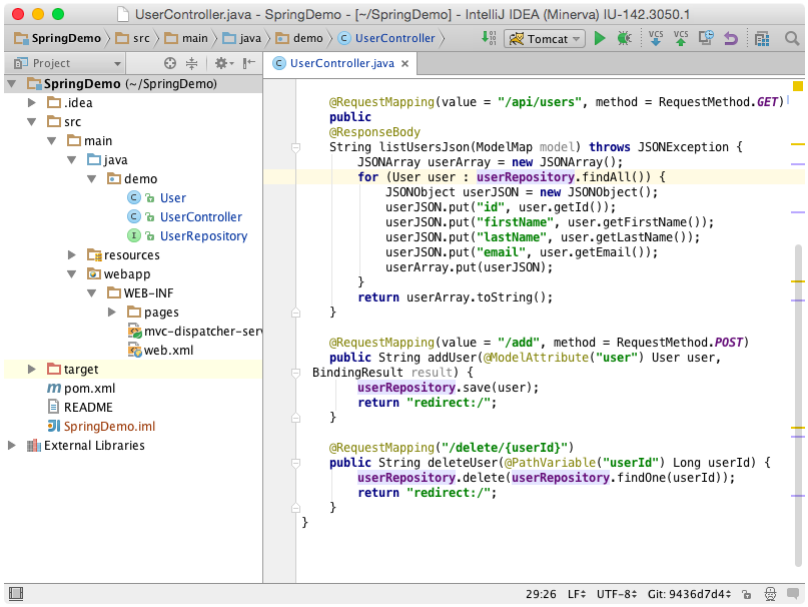
wish you a good luck with finding your way through Intellij IDEA!

Discover IntelliJ IDEA

This guide is intended to help you become more productive with IntelliJ IDEA, and provides an overview of the most important features, together with tips, tricks, and the hottest shortcuts.

User interface

The IntelliJ IDEA Editor is special in a number of ways, most notable being is that you can invoke almost any IDE feature without leaving it, which allows you to organize a layout where you have more screen space because auxiliary controls like toolbars and windows are hidden.



Accessing a tool window via its shortcut moves the input focus to it, so you can use all keyboard commands in its context. When you need to go back to the editor, press `Escape`.

Below is a list of shortcuts that invoke the tool windows you will most often need:

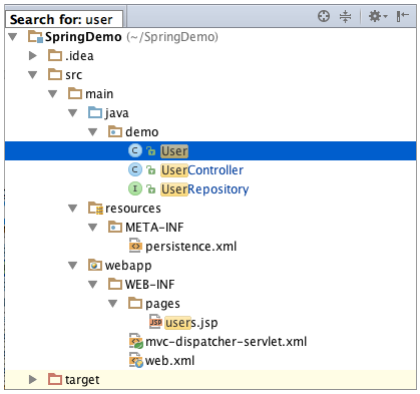
Tool Window	Shortcut
Project	`Alt+1`
Version Control	`Alt+9`
Run	`Alt+4`
Debug	`Alt+5`
Terminal	`Alt+F12`
Editor	`Escape`

When you want to focus on the code, try the Distraction Free Mode . It removes all toolbars, tool windows, and editor tabs. To switch to this mode, on the main menu select View | Enter Distraction Free Mode.

An alternative to the Distraction Free Mode may be hiding all tool windows by pressing `Ctrl+Shift+F12`. You can restore the layout to its default by pressing this shortcut once again.

The Navigation Bar is a compact alternative to the Project tool window . To access the Navigation Bar, press `Alt+Home`.

Most components in IntelliJ IDEA (both tool windows and popups) provide speed search . This feature allows you to filter a list, or navigate to a particular item by using a search query.



Tip When you don't know the shortcut for an action, try using the Find action feature by pressing Ctrl+Shift+A. Start typing to find an action by its name, see its shortcut or call it.

Refer to Guided Tour around the User Interface , Editor . and Tool Windows .

Code completion

When you access Basic Completion by pressing Ctrl+Space, you get basic suggestions for variables, types, methods, expressions, and so on. When you call Basic Completion twice, it shows you more results, including private members and non-imported static members.

The Smart Completion feature is aware of the expected type and data flow, and offers the options relevant to the context. To call Smart Completion, press Ctrl+Shift+Space. When you call Smart Completion twice, it shows you more results, including chains.

Tip To overwrite the identifier at the caret, instead of just inserting the suggestion, press Tab. This is helpful if you're editing part of an identifier, such as a file name.

To let IntelliJ IDEA complete a statement for you, press Ctrl+Shift+Enter. Statement Completion will automatically add the missing parentheses, brackets, braces and the necessary formatting.

If you want to see the suggested parameters for any method or constructor, press Ctrl+P. IntelliJ IDEA shows the parameter info for each overloaded method or constructor, and highlights the best match for the parameters already typed.

The feature lets you transform an already typed expression to another one based on the postfix you type after a period, the expression type, and its context.

Editor basics

Since in IntelliJ IDEA you can undo refactorings and revert changes from Local History , it makes no sense to ask you to save your changes every time.

The most useful Editor shortcuts are:

Action	Description
Move the current line of code	Ctrl+Shift+Up Ctrl+Shift+Down
Duplicate a line of code	Ctrl+D
Remove a line of code	Ctrl+Y
Comment or uncomment a line of code	Ctrl+Slash
Comment a block of code	Ctrl+Shift+Slash
Find in the currently opened file	Ctrl+F
Find and replace in the current file	Ctrl+R
Next occurrence	F3
Previous occurrence	Shift+F3
Navigate between opened tabs	Alt+Right Alt+Left
Navigate back/forward	Ctrl+Alt+Left Ctrl+Alt+Right
Expand or collapse a code block in the editor	Ctrl+NumPad Plus Ctrl+NumPad -
Create new...	Alt+Insert
Surround with	Ctrl+Alt+T
Highlight usages of a symbol	Ctrl+F7

To expand a selection based on grammar, press Ctrl+W. To shrink it, press Ctrl+Shift+W.

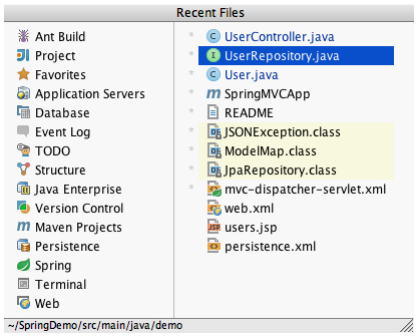
IntelliJ IDEA can select more than one piece of code at a time. You can select/deselect any piece of code via Alt+J, or by clicking a code selection and pressing Shift+Alt+J.

Refer to Editor .

Navigation

Recent files

Most of the time you work with a finite set of files, and need to switch between them quickly. A real time-saver here is an action called Recent Files invoked by pressing Ctrl+E. By default, the focus is on the last accessed file. Note that you can also open any tool window through this action:



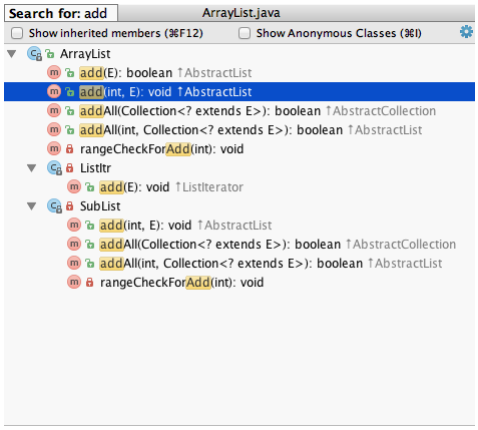
Navigate to Class is available by pressing Ctrl+N and supports sophisticated expressions, including camel humps, paths, line navigate to, middle name matching, and many more. If you call it twice, it shows you the results out of the project classes.

Navigate to File works similarly by pressing `Ctrl+Shift+N`, but is used for files and folders. To navigate to a folder, end your expression with the `Slash` character.

Navigate to Symbol is available by pressing `Ctrl+Shift+Alt+N` and allows you to find a method or a field by its name.

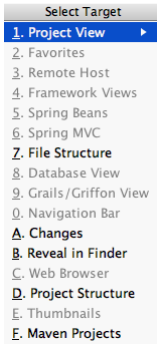
Structure

When you are not switching between files, you are most probably navigating within a file. The simplest way to do it is to press `Ctrl+F12`. The popup shows you the structure of a file, and allows you to quickly navigate to any of them:



Select in

If you need to open a file in a particular tool window (or Finder/Explorer), you can do so via the Select In action by pressing `Alt+F1`:



Navigation shortcuts include:

Action	Shortcut
Search everywhere	`Double Shift`
Navigate to class	`Ctrl+N`
Navigate to file	`Ctrl+Shift+N`
Navigate to symbol	`Ctrl+Shift+Alt+N`
Recent files	`Ctrl+E`
File structure	`Ctrl+F12`
Select in	`Alt+F1`
Navigate to declaration	`Ctrl+B`
Navigate to type hierarchy	`Ctrl+H`
Show UML popup	`Ctrl+Alt+U`

Refer to Navigating Through the Source Code .

Quick pop-ups

Quick Pop-ups are helpful for checking additional information related to the symbol at the caret. Below is a list of pop-ups you should know if you want to be more productive:

Action	Shortcut
Documentation	`Ctrl+Q`
Quick definition	`Ctrl+Shift+I`
Show usages	`Ctrl+Alt+F7`
Show implementation	`Ctrl+Alt+B`

Tip Quick Pop-ups are available for symbols in the editor; however, they are also available for items in any other list via the same shortcuts.

Refactoring basics

IntelliJ IDEA offers a comprehensive set of automated code refactorings that lead to significant productivity gains when used correctly. Firstly, don't bother selecting anything before you apply a refactoring. IntelliJ IDEA is smart enough to figure out what statement you're going to refactor, and only asks for confirmation if there are several possible choices.

Tip To undo the last refactoring, switch the focus to the Project tool window and press `Ctrl+Z`.

Action	Shortcut
Rename	`Shift+F6`
Extract variable	`Ctrl+Alt+V`
Extract field	`Ctrl+Alt+F`
Extract a constant	`Ctrl+Alt+C`
Extract a method	`Ctrl+Alt+M`
Extract a parameter	`Ctrl+Alt+P`
Inline	`Ctrl+Alt+N`
Copy	`F5`
Move	`F6`
Refactor this	`Ctrl+Shift+Alt+T`

Tip A real time-saver is the ability to extract part of a string expression with the help of the Extract refactorings. Just select a string fragment and apply a refactoring to replace all of the selected fragment usages with the introduced constant or variable.

Refer to Refactoring Source Code.

Finding usages

Find Usages helps you quickly find all pieces of code referencing the symbol at the caret (cursor), no matter if the symbol is a class, method, field, parameter, or another statement. Just press `Alt+F7` and get a list of references grouped by usage type, module, and file.

If you want to set custom options for the Find Usages algorithm, press `Ctrl+Shift+Alt+F7`, or click the first button on the right panel with search results.

If what you're looking for is plain text, use Find in Path by pressing `Ctrl+Shift+F`.

Refer to Finding Usages.

Inspections

Inspections are built-in static code analysis tools that help you find probable bugs, locate dead code, detect performance issues, and improve the overall code structure.

Most inspections not only tell you where a problem is, but also provide quick fixes to deal with it right away. Press `Alt+Enter` to choose a quick fix.

Tip The editor lets you quickly navigate between the highlighted problems via keyboard shortcuts. Press `F2` to go to the next problem, and `Shift+F2` to go to the previous one.

Inspections that are too complex to be run on-the-fly are available when you perform code analysis for the entire project. You can do this in one of the following two ways: by selecting Analyze | Inspect Code from the main menu, or by selecting Analyze | Run Inspection by Name to run an inspection by its name.

Note that while inspections provide quick-fixes for code that has potential problems, intentions help you apply automatic changes to code that is correct. To get a list of intentions applicable to the code at the caret, press `Alt+Enter`.

Refer to Code Inspection.

Code style and formatting

IntelliJ IDEA automatically applies a code style you've configured in the Code Style settings as you edit, and in most cases you don't need to call the Reformat Code action explicitly.

Useful formatting shortcuts:

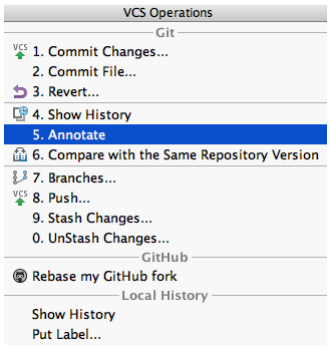
Action	Shortcut
Reformat code	`Ctrl+Alt+L`
Auto-indent lines	`Ctrl+Alt+I`
Optimize imports	`Ctrl+Alt+O`

Note that by default, IntelliJ IDEA uses regular spaces for indents instead of tabs. If you have files with lots of indents, you may want to optimize their size by enabling the Use tab character option in the Java code style settings.

Version control basics

To check out a project from a Version Control System (VCS), click Checkout from Version Control on the Welcome Screen, or in the main VCS menu.

To quickly perform a VCS operation on the current file, directory, or an entire project, use the VCS operations popup by pressing `Alt+Back Quote`



Once you've configured the VCS settings, you'll see the Version Control tool window . You can switch to it anytime by pressing `Alt+9` .

The Local Changes tab of the Version Control tool window shows your local changes: both staged and unstaged.

Useful VCS shortcuts:

Action	Shortcut
Version Control tool window	`Alt+9`
VCS operations popup	`Alt+Back Quote`
Commit changes	`Ctrl+K`
Update project	`Ctrl+T`
Push commits	`Ctrl+Shift+K`

Tip Annotation (available from both the quick list, the main and the context menus) allows you to see who and when changed a line of code for any file.

Branches

To perform an operation on a branch, either select Branches from the VCS main or context menu, the VCS operations popup, or the widget on the right of the status bar.

Note that for multiple repositories, IntelliJ IDEA performs all VCS operations on all branches simultaneously, so you don't need to switch between them manually.

Shelves , stashes and patches help you when you need to store some of the local changes without committing them to the repository. You can then switch to the repository versions of the files, and then come back to your changes later.

Make

By default, IntelliJ IDEA doesn't automatically compile projects on saving. To compile a project, select Build | Make Project from the main menu and press `Ctrl+F9` .

Refer to Compiling Applications .

Running and debugging

Once you've created a Run/Debug configuration by selecting Run | Edit Configurations from the main menu, you are able to run and debug your code.

Action	Shortcut
Run	`Shift+F10`
Debug	`Shift+F9`

When in the debug mode, you can evaluate any expression by using the Evaluate expression tool, which is accessed by pressing `Alt+F8` . This tool provides code completion in the same way as in the editor, so it's easy to enter any expression.

Sometimes, you may want to step into a particular method, but not the first one which will be invoked. In this case, use Smart step into by pressing `Shift+F7` to choose a particular method.

Action	Shortcut
Toggle breakpoint	`Ctrl+F8`
Step into	`F7`
Smart step into	`Shift+F7`
Step over	`F8`
Step out	`Shift+F8`
Resume	`F9`
Evaluate expression	`Alt+F8`

If you want to "rewind" while debugging, you can do it via the Drop Frame action. This is particularly helpful if you mistakenly stepped too far. This will no revert the global state of your application, but will at least let you revert to a previous stack frame.

Tip Any breakpoint can be quickly disabled by clicking on the gutter while holding `Alt` . To change breakpoint details (e.g. conditions), press `Ctrl+Shift+F8` .

Reloading changes and hot swapping

Sometimes, you need to insert minor changes into your code without shutting down the process. Since the Java VM has a HotSwap feature, IntelliJ IDEA handles these

cases automatically when you call Make.

Application servers

To deploy your application to a server:

1. Configure your artifacts by selecting File | Project Structure | Artifacts (done automatically for Maven and Gradle projects).
2. Configure an application server by clicking the Application Servers page of the Settings/Preferences dialog.
3. Configure a run configuration by selecting Run | Edit Configurations, then specify the artifacts to deploy and the server to deploy them to.

You can always tell IntelliJ IDEA to build/rebuild your artifacts (once they have been configured) by selecting Build | Build Artifacts.

Tip When you need to apply changes in the code to a running application, in addition to Make, you can use the Update action by pressing `Ctrl+F10`. This action is only available for the Exploded artifact type. Based on your choice, it can update resources or update classes and resources. When the Update action is applied in the Debug mode, it uses HotSwap; otherwise, it uses Hot redeployment.

Working with build tools (Maven/Gradle)

Once you've imported/created your Maven/Gradle project, you are free to edit its `pom.xml` or `build.gradle` files directly in the editor. Any changes to the underlying build configuration will eventually need to be synced with the project model in IntelliJ IDEA.

If you want the IDE to synchronize your changes immediately, do the following:

- For `pom.xml` enable the Import Maven projects automatically option in File | Settings | Build, Execution, Deployment | Build Tools | Maven | Importing (Windows and Linux) or IntelliJ IDEA | Preferences | Build, Execution, Deployment | Build Tools | Maven | Importing (OS X).
- For `build.gradle`, enable the Use auto-import option in Build, Execution, Deployment | Build Tools | Gradle of the Settings/Preferences dialog.

For manual synchronization, use the corresponding action on the Maven/Gradle tool window toolbar: .

Note that any goal or task can be attached to be run before a run configuration.

Refer to Build Tools .

Migrating to Eclipse or NetBeans

If you are considering the possibility to migrate from Eclipse or NetBeans to IntelliJ IDEA, refer to the corresponding migration guides .

What's next

We strongly advise you to read the documentation. Also you might find it useful to refer to the Java tutorials under Java SE , and also to the tutorial on Java EE .

Keyboard Shortcuts You Cannot Miss

IntelliJ IDEA, as a keyboard-centric IDE, suggests keyboard shortcuts for most of its actions. In this topic, you can find a short list of the most indispensable of them, to make your first steps with IntelliJ IDEA easy.

Tip IntelliJ IDEA also provides the default keymap reference in the pdf format. To view the keymap reference, choose Help | Keymap Reference from the main menu.

See the detailed list of default keyboard shortcuts in the Keyboard Shortcuts Reference and learn how to customize your preferred keymap in the Configuring Keyboard Shortcuts section.

Action	Shortcut
Find action by name	Ctrl+Shift+A
Show the list of available intention actions .	Alt+Enter
Switch between views (Project , Structure , etc.).	Alt+F1
Switch between the tool windows and files opened in the editor.	Ctrl+Tab
Show the Navigation bar .	Alt+Home
Insert a live template .	Ctrl+J
Surround with a live template .	Ctrl+Alt+J
Edit an item from the Project or another tree view .	F4
Comment or uncomment a line or fragment of code with the line or block comment.	Ctrl+Slash Ctrl+Shift+Slash
Find class or file by name .	Ctrl+N Ctrl+Shift+N
Duplicate the current line or selection .	Ctrl+D
Incremental expression selection .	Ctrl+W and Ctrl+Shift+W
Find/replace text string in the current file .	Ctrl+F Ctrl+R
Find/replace text in the project or in the specified directory	Ctrl+Shift+F Ctrl+Shift+R
Search everywhere.	Double-press Shift
Quick view the usages of the selected symbol .	Ctrl+Shift+F7
Expand or collapse a code fragment in the editor .	Ctrl+NumPad Plus Ctrl+NumPad -
Invoke code completion .	Ctrl+Space
Smart statement completion .	Ctrl+Shift+Enter
Smart completion	Ctrl+Shift+Space
Show the list of available refactorings (Refactor This).	Ctrl+Shift+Alt+T

The complete keymap reference is available from the main menu (Help | Keymap Reference).

IntelliJ IDEA Pro Tips

Overview

This guide targets IntelliJ IDEA users who are already familiar with its basic features and would like to learn more. If you're relatively new to IntelliJ IDEA, we recommend that you read the Discover IntelliJ IDEA guide before delving into this one.

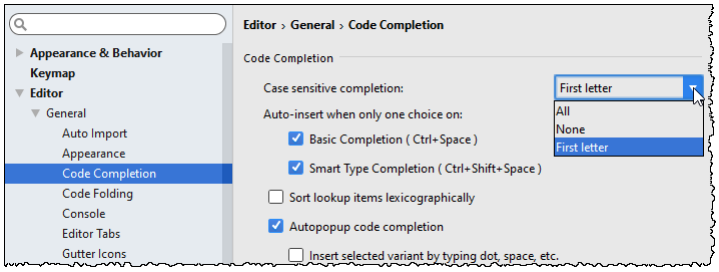
Coding assistance

Type Info

If you want more information about a symbol at caret, e.g. where it comes from or what its type is, the Quick Documentation is your friend. Press `Ctrl+Q` to invoke it and you will see a popup with these details. If you don't need the full info, then use the Type Info action instead: it only shows the type of a selected expression, but doesn't take up that much of screen space.

Code completion case sensitivity

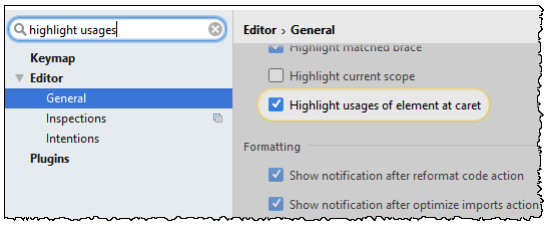
By default IntelliJ IDEA code completion case sensitivity only affects the first letter you type. This strategy can be changed in the Settings/Preferences dialog, Editor | General | Code Completion, so you can make to either make the IDE sensitive to all letters or make it insensitive to the case at all, based on what better fits your preferences.



Hot tip: Here you can also turn off the Autopopup code completion option. This makes sense if you want the code completion popup to show up only when you explicitly call it.

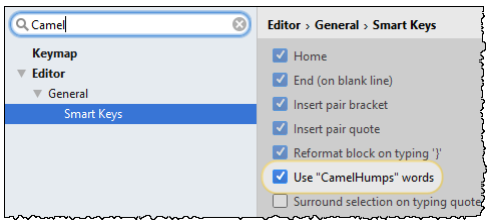
Disable highlighting usages of element at caret

Talking about the defaults you may want to change after learning IntelliJ IDEA better, we can't miss the Highlight usages of element at caret option in the Editor | General page of the Settings/Preferences dialog. If you know the `Ctrl+Shift+F7` shortcut and don't like the highlighting in the editor to appear and disappear each time you simply move the caret, you don't need this option.



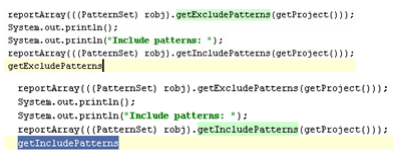
CamelHumps

By default, when you select anything in the editor, IntelliJ IDEA isn't sensitive to the case of the words. If you prefer to select words according to CamelCase, e.g. instead of selecting the whole word, select a part of it, you can enable this in Editor | General | Smart Keys of the Settings/Preferences dialog.



Hippie completion

IntelliJ IDEA provides Basic completion via `Ctrl+Space`, Smart completion via `Ctrl+Shift+Space`, and Statement completion via `Ctrl+Shift+Enter`. All these features are based on actual understanding of the code structure. However, sometimes you may need a more trivial, yet flexible logic that would suggest the words used earlier in the current file or even project regardless to their context. This feature is called Hippie completion and is available via `Alt+Slash`.



Refactorings

Undo refactorings

With IntelliJ IDEA you don't need to worry about consequences when refactoring code, because you can always undo anything by invoking Undo via the convenient `Ctrl+Z` shortcut.

Extract string fragments

IntelliJ IDEA is capable of refactoring not only executable code, but also string literals. Select any fragment of a string, call Extract variable/constant/field/parameter to extract it as a constant and replace its usages throughout the code.

Type migration

When you refactor, you usually rename symbols, or extract and move statements in the code. However, there's more to refactoring than just that. For example, Type Migration (available via `Ctrl+Shift+F6`) lets you change the type for a variable, field, parameter or a method's return value (`int → String` , `int → Long` , etc), update the dependant code, and resolve possible conflicts.

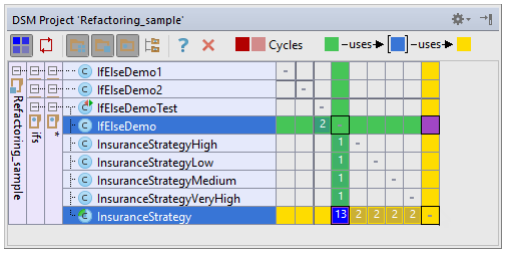
Invert boolean

If IntelliJ IDEA can automate type migration, why not do the same with semantics? To invert all usages of a boolean symbol, just use the Invert Boolean refactoring .

Code analysis

Dependency structure matrix

IntelliJ IDEA lets you analyze how tightly components in your code depend on each other, and you need to keep an eye on that because when there's too much dependencies, it's likely to cause various problems . Dependency Structure Matrix action (available via the Analyze menu) will help you visualize and explore dependencies between modules, packages and classes.

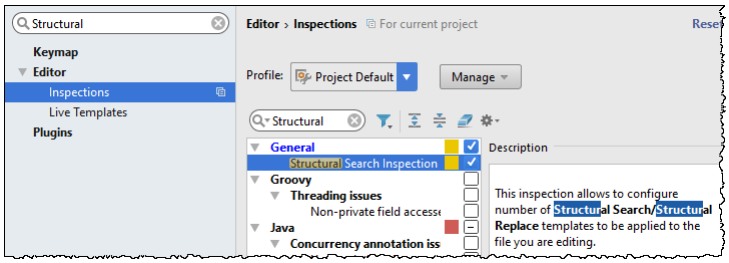


Despite its complex looks, it's a very easy-to-use tool. Just select a class or package and see where it's used and what it uses.

Structural search and replace

Structural Search and Replace, or SSR , is quite powerful (after you learn to use it right), and can be used for static code analysis and refactoring automation. In a nutshell, it lets you search for specific patterns in your code and replace them with parametrized templates. For that, it's equipped with its own language for defining code patterns that is described in more detail in this article .

To access this feature, use the Edit | Find | Search/Replace Structurally.... If you want to create your templates or patterns, go to Settings/Preferences dialog, click the page Editor | Inspections , and enable Structural Search Inspection under the General node:



User interface

Disable breadcrumbs and tag tree highlighting

If you work with lots of HTML and XML and would like to avoid unnecessary distraction, you may want to disable breadcrumbs and tag tree highlighting in Editor | General | Appearance .

Disable unnecessary gutter icons

Gutter, the leftmost editor column, typically displays useful information related to the code you're editing. If you feel that sometimes it's just too much, you can configure what you want to see in the Settings/Preferences dialog: Editor | General | Gutter Icons .

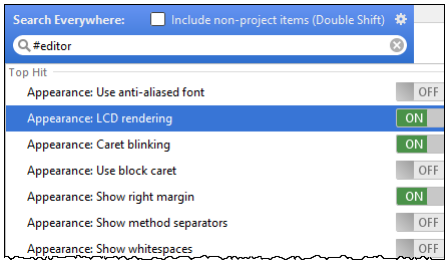
Disable annoying intention light bulb

One more thing that might be annoying is the intention bulb that appears in the editor every time there is an intention available at the caret. Disabling that is a little bit more difficult: you need to manually edit your `<IntelliJ IDEA preferences folder>/options/editor.xml` , and add the following line:

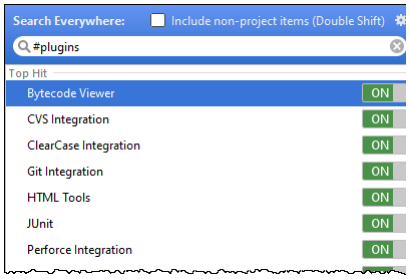
```
<option name="SHOW_INTENTION_BULB"
                    value="false" />
```

Using from Search everywhere

With Search Everywhere you can find arbitrary text fragments literally everywhere: in the code, libraries, parts of the UI, settings (by prepending the settings name with `#`), or even action names. If you're using this feature a lot, it's worth knowing that you can access IntelliJ IDEA settings by just pressing `Enter` right in its popup. For example, here we're accessing the editor settings:

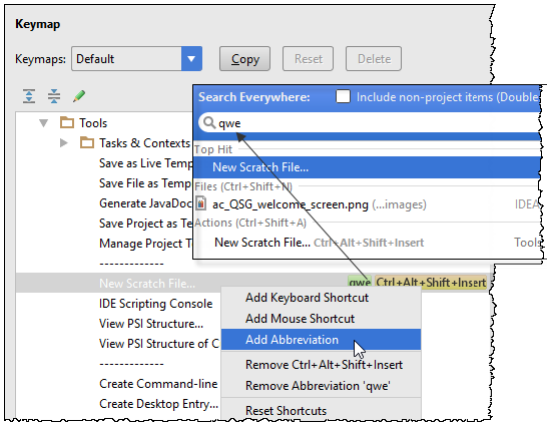


If you start your search query with `#plugins`, you'll be able to turn them on and off:



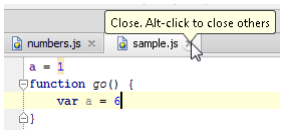
Other tags include `#appearance`, `#system`, `#inspections`, `#registry`, `#intentions`, `#templates`, and `#vcs`.

Another interesting fact is that Search Everywhere supports abbreviations. You can use Keymap page of the Settings/Preferences dialog to assign a short text to any action, and then have this action called from Search Everywhere by entering this text:



Hide editor tabs

When you need to close all editor tabs except the current one, click the close icon on the current tab holding `Alt`:



If you don't want to see the editor tabs at all, go to the Editor Tabs page of the Editor Settings/Preferences and under the Placement drop-down select None.

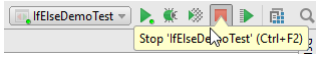
Open file in new window

A feature that is not that easy to find, yet comes in handy, is opening a file in a new window by selecting it in the Project Tool Window and clicking `Shift+Enter`.

Add stop and resume buttons to the toolbar

It might be convenient to add Stop and Resume buttons to the toolbar of the Navigation Bar. You can do it via the Appearance and Behavior | Menus and Toolbars page of the Settings/Preferences dialog.

If you prefer to use the mouse rather than keyboard shortcuts, this way you won't need to open the Debug tool window to manage your current debugging session.



Editor

Compare with clipboard

IntelliJ IDEA has a built-in Diff viewers for code, jar files, revisions and even images. To invoke it, select any pair of files and press `Ctrl+D`.

If you have selected a single file, the IDE will prompt you to select the one to compare to. To quickly compare active editor with Clipboard, choose View | Compare with Clipboard.

Paste from history

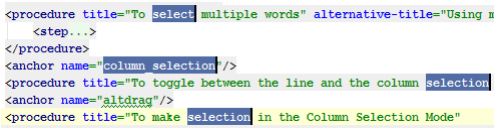
Speaking of Clipboard, IntelliJ IDEA keeps track of everything you put there. Anytime you want to paste one of the previously copied items, press `Ctrl+Shift+V`.

Multiple selections

Multiple selection is a relatively new, very powerful editor feature, which lets you quickly select and edit multiple (adjacent or not) pieces of code at once.

In a nutshell, here's what happens. You either start with pressing `Alt+J` (and then IntelliJ IDEA selects a symbol at caret), or you can just select something as you normally would.

Then, press `Alt+J` and IntelliJ IDEA will search the current file forward until it finds a matching piece of text, which it adds to the selection. You can press `Alt+J` again to go forward, or `Shift+Alt+J` to go back, but note that when search reaches the end of file, it will start over from the beginning of the file.



After selection is complete, you can start editing all the fragments as if they were one.

Hot tip: One more way to clone caret is to press `Ctrl` (`Alt` for OS X) twice, and then move the caret up or down with arrows or simply with mouse.

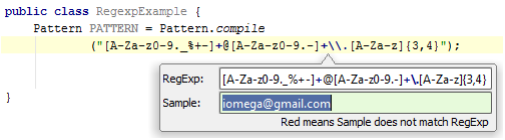
Emmet

In case you didn't know, Emmet is a great way to write HTML, XML and CSS code. IntelliJ IDEA supports it out of the box : simply write an Emmet expression and press `Tab` to expand it.

Use the Emmet preview action (which is available vis Find Action or Search Everywhere - so make sure to assign it to a handy shortcut) to see a preview of the resulting code.

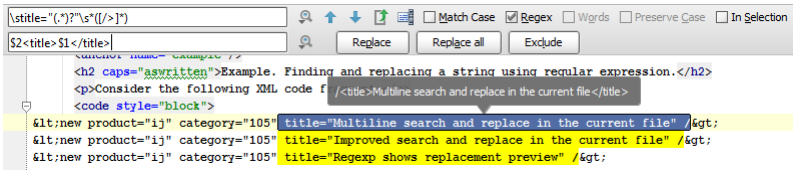
Regex

Regular expressions are powerful and widely used, but sometimes it's just too hard to write them properly. IntelliJ IDEA will help you check any Regex in your code: just place caret in it and press `Alt+Enter` to use the Check Regex intention:



Find and replace with Regex groups

Another place where IntelliJ IDEA helps with Regex is the Find and Replace feature . It's worth knowing that it supports captured groups in replacement expressions.



Find and replace also lets you exclude comments and literals from search: to do that, use the Gear icon .

Bytecode viewer

Sometimes seeing the actual bytecode your program generates is very insightful .

In IntelliJ IDEA you can always do that via View | Show Bytecode.

Version control

Amend changes

In the Commit Changes dialog IntelliJ IDEA offers to perform a variety of operations. One of them is Amend commit, which is useful when you want to change your last commit and join your current change to it.

Shelves and patches

Shelves is an IDE feature similar to Git Stash , but that works for all VCS: it helps when you need to pause your current work and pull something from the repository to fix it asap, and then resume working on whatever it was you were working on. This feature takes care of locally changed files without committing them, so no more lost changes or hastily made merge commits.

Refer to the page Git-Stach and to the section Stashing and Unstashing for more detail.

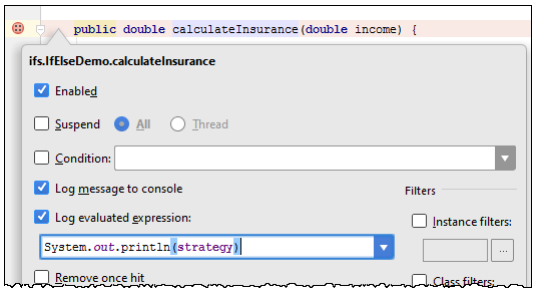
Patches allow you to save a set of changes to a text file that can be transferred via email (or any other ancient medium), and then applied to the code. It's super helpful when you really need to commit something after your plane crash landed on a desert island, or you somehow else got yourself in a situation without reliable broadband connection.

Refer to the section Using Patches for more detail.

Debugging

Action, or method breakpoints

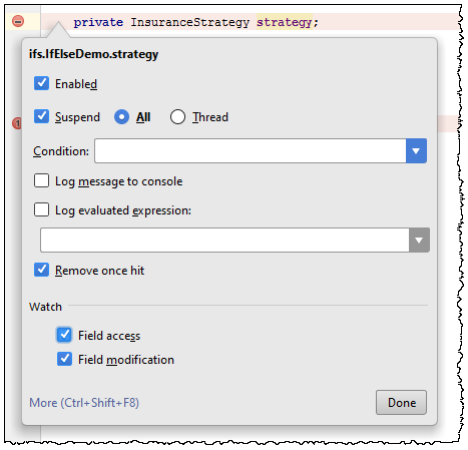
Sometimes you may want to evaluate something at a particular line of code without actually making a stop. You can do that by using a Method breakpoint . To create one, just click the gutter holding `Shift` .



This way you can print any expression to the output without changing the code. This is especially useful when you debug libraries or a remote application.

Field breakpoints or field watchpoints

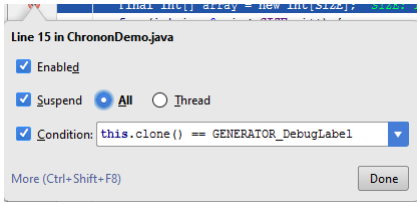
In addition to the action breakpoints mentioned above, you can also use Field watchpoints . This breakpoint will stop execution when a field associated with it is accessed. To create field watchpoints, just click the gutter holding `Alt` (`Ctrl+Cmd` for OS X).



Object markers

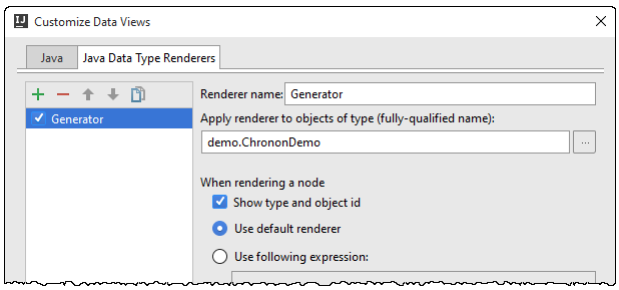
When you're debugging an application, IntelliJ IDEA lets you mark particular instances of arbitrary objects with colored labels for easier identification via the Mark Object action (available in the Evaluate Expression , Variables or Watches views.)

And if you have any instance marked with a label, you can use it in the condition expression as well:



Custom data renderers

Evaluate Expression , Variables , Watches and inline debugger all use a standard way to render variable values, mostly based on `toString` implementation of classes. Not everyone knows that you can define your own custom renderers for any class. For that, select Customize Data Views from the context menu in the Debug tool window.



This is especially useful when some of the classes in the libraries you're using do not provide a meaningful a `toString` implementation–so you can define it yourself outside of the library.

Drop frame

In case you want to "go back in time" while debugging you can do it via the Drop Frame action. This is a great help if you mistakenly stepped too far. This will not revert the global state of your application but at least will get you back by stack of frames.

Force return

The way around, if you want to jump to the future, and force the return from the current method without executing any more instructions from it, use the Force Return action (to invoke it, press `Ctrl+Shift+A` and type the action name). If the method returns a value, you'll have to specify it.

DCEVM

Sometimes when you're making quick changes to code, you want to immediately see how they will behave in a working application. Unfortunately, the Java HotSwap VM has lots of limitations: you can't, say, add a new method or a field to a class and perform the hot swapping; the only thing you can actually change during the hot swapping is the method bodies.

Refer to the sections Reloading Classes and HotSwap for details.

Luckily, there is a way to amend this situation with the new open-source project Dynamic Code Evolution VM, a modification of Java HotSwap VM with unlimited support for reloading classes at runtime.

Using it in IntelliJ IDEA is easy with the dedicated plugin . When you enable the plugin, the IDE will offer you to download DCEVM JRE for your environment. Then you'll have to choose it in the list of alternative JREs.

Update application

If you are running your application on an application server (e.g. Tomcat, JBoss, etc), can you reload changed classes and resources using the Update application action via `Ctrl+F10`.

Refer to the section Updating Applications on Application Servers for details.

Tools

External tools

IntelliJ IDEA has many developer tools integrated and working out of the box. If a tool you need is not integrated, but you'd like to use it via a shortcut, go to Settings/Preferences | Tools | External Tools , and configure how to run this tool. Then you'll be able to run this tool via the Tools | External Tools main menu.

Refer to the section Configuring Third-Party Tools .

Getting Help

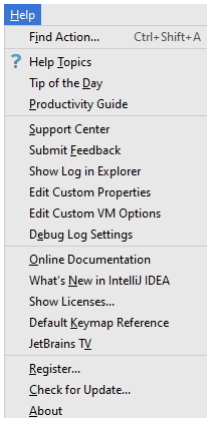
There are a lot of ways to obtain information about IntelliJ IDEA and its features, report issues and get support.

In this section:

- Getting Help
  - Overview
  - Menu commands
- Using Help Topics
- Using Tips of the Day
- Using Online Resources
- Using Productivity Guide
- Reporting Issues and Sharing Your Feedback
- Keymap Reference

Overview

Use the Help menu commands:



Menu commands

Menu item	Keyboard shortcut	Description
Find Action	`Ctrl+Shift+A`	Choose this command to invoke an action by its name .
Demos and Screencasts		Choose this command to open the IntelliJ IDEA YouTube page.
Help Topics		Choose this command to show the help topics. Refer to the section Using Help Topics .
Tip of the Day		Choose this command to show an arbitrary tip. Refer to the section Using Tips of the Day .
Productivity Guide		Choose this command to show productivity guide .
Support Center		Choose this command to open JetBrains Support page.
Submit Feedback		Choose this command to report your overall impression of IntelliJ IDEA to the support service. Refer to the section Reporting Issues and Sharing Your Feedback .
Show Log in Explorer/Finder		Choose this command to find IntelliJ IDEA's log. Refer to the section Reporting Issues and Sharing Your Feedback for details.
Edit Custom Properties		Choose this command to open the custom file `idea.properties` , located under the user home. If this file does not exist, IntelliJ IDEA suggests to create it. Refer to the section Tuning IntelliJ IDEA for details.
Edit Custom VM Options		Choose this command to open the custom file `*.vmoptions` , located under the user home. If this file does not exist, IntelliJ IDEA suggests to create it. Refer to the section Tuning IntelliJ IDEA for details.
Debug Log Settings		Choose this command to change logging level for a category. Choosing this command leads to opening the Custom Debug Log Configuration dialog box, where you have to type the log categories names, separated with new lines. Refer to the section Reporting Issues and Sharing Your Feedback .
Keymap Reference		Choose this command to open the default keymap in PDF format. Refer to the section Keymap Reference for details.
What's New in IntelliJ IDEA		Choose this command to open the What's New page .
Show Licences		Choose this command to show the legal information.
Diagnostics		This command and the items below it are available only when the Performance plugin is installed and activated. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Use the commands under this menu item to automatically capture snapshots and collect other data related to performance and send this data to JetBrains Support . When you click Diagnostics, a drop-down menu with the following items is displayed: – Enable CPU Profiler: choose this option to activate the CPU profiler and capture CPU snapshots. The CPU profiles remains active and the option not available till you restart IntelliJ IDEA.

	– Change Memory Settings: choose this option to edit the `-Xmx` setting and thus increase the heap size (the amount of available memory), see Tuning IntelliJ IDEA for details. – Execute Performance Script: choose this option to run special predefined scenarios with the CPU profiler enabled. After a scenario is executed it generates a CPU snapshot that can be used by the support team to analyze your performance issue. – Profile Indexing: when you choose this option, CPU profiler is active and captures a CPU snapshot from the moment you reopen the current project till the moment indexing is completed. The snapshot can be used by the support team to analyze a performance issue that arize during indexing. – Show Performance Summary: choose this option to collect most relevant information that can affect performance and at the same time does not reveal any private info about the current project and the current file. Learn more at Performance testing plugin in PhpStorm
Settings Summary	Choose this command to collect the IntelliJ IDEA settings and show all sorted by category so it is easy to locate the settings that are related with a specific problem (debugger, or deployment, or something else does not work) and send this summary to JetBrains Support . To open a request form, click Open Support Request in the dialog box that opens. This command is available only when the Performance plugin is installed and activated. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
New Support Request	Choose this command to open the support ticket page. Fill out the fields (in particular, use the Settings Summary copy), and click Submit.
JetBrains TV	Choose this command to open JetBrains TV page .
Register	Choose this command to register IntelliJ IDEA.
Check for Update	Choose this command to obtain information about the current version, and the availability of newer versions of IntelliJ IDEA. Refer to Updates page. This command in available on Windows/Linux. On Mac OS it appears on the IntelliJ IDEA menu.
About	Choose this command to obtain information about the current version of IntelliJ IDEA, current build, etc. Press `Escape` to close the pop-up window. This command in available on Windows/Linux. On Mac OS it appears on the IntelliJ IDEA menu.

Using Help Topics

In this section:

- Documentation structure
- Built-in documentation

  This type of documentation is available for Ultimate edition only.
- Online documentation

  This type of documentation is available for both Ultimate and Community editions.

Documentation structure

IntelliJ IDEA documentation has the following structure:

Meet IntelliJ IDEA

This part contains system requirements and installation information, quick start guide that helps you get a grip of IntelliJ IDEA, the section Discover IntelliJ IDEA that outlines the main features of IntelliJ IDEA, the section First Steps with the tutorials that help you perform the most basic actions - create and run your first application.

Migration Guides

This part is intended for Eclipse and NetBeans users, who are about to move to IntelliJ IDEA.

How to

This contains information related to the platform features (such as, for example, using the IntelliJ IDEA editor , tool windows , or version control), and language- and framework-specific guidelines .

Reference

This part contains the miscellaneous information, which includes the basic concepts, or essentials , dialogs reference , icons reference , and more.

Using built-in documentation

Built-in documentation enables you to browse through the topics using the table of contents, find occurrences in the Search tab, or use the detailed Index that contains all keywords from all topics.

To bring up help contents, do one of the following

- On the main menu, choose Help | Help Topics.
- Press F1.
- Click Help button, if it is available.

To find a particular piece of information

1. Click the Search tab of the help viewer.
2. In the search field, type the search string and press Enter.
3. In the list of topics that contain occurrences of the search string, select the desired one. Occurrences are highlighted in the right pane of the help viewer.

To find a keyword in the Index tab

1. Click the Index tab of the help viewer.
2. In the search field, type the desired keyword and press Enter. The caret rests at the first occurrence of the keyword. Every time you press Enter, the caret moves to the next occurrence of the keyword. To see the information about a keyword, select one of its sub-entries.

Online documentation

On the IntelliJ IDEA site, find online documentation:

IntelliJ IDEA web help

The online version makes it possible to find entries in the table of contents, browse documentation with the table of contents, rate topics and express your opinion. The layout of online documentation consists of:

Table of contents pane

This pane shows the table of contents.

- Use this pane to browse through the topics.
- Click button to show or hide this pane.
- If for some reason your browser fails to show actual table of contents, refresh the page.

Topics pane

This pane shows the topic that is currently selected in the table of contents.

- In the Keymap drop-down list, choose the platform you want to view keyboard shortcuts in (Windows/Linux, OS X etc.)

Finding a piece of information in the table of contents

1. Switch to the Table of contents pane.

2. In the Search IntelliJ IDEA help field, type your query.

3. Press Enter.
   The table contents shrinks to show the search results. Click the desired entry to show the corresponding page in the Topic pane.

Using Tips of the Day

Tips of the Day provide a collection of useful and interesting hints. They show up every time you start IntelliJ IDEA.

To show Tips of the Day

– Choose Help | Tip of the Day on the main menu.

To navigate through the collection of tips

– Use the Previous and Next buttons.

To suppress Tips of the Day

– In the Tips of the Day window, clear the check box Show Tips on Startup .

Using Online Resources

If built-in documentation fails to answer your questions, you can find more information on the web. The following resources are available:

- Official JetBrains home page
- IntelliJ IDEA community
- Blog
- Online version of documentation contains the latest updates.
- Collection of articles, white papers etc.

Finally, do not miss the JetBrains TV (Help | JetBrains TV). On the JetBrains TV page, choose the IntelliJ IDEA channel, and watch screencasts.

Using Productivity Guide

IntelliJ IDEA smartly analyzes the features you use most often during your development sessions, and reminds you of the features you might have missed.

The Productivity Guide dialog, available in Help | Productivity Guide, displays the list of features with usage statistics and tips.

Besides analysing your personal usage of features, you can discover similar features that you've never used. One of the ways to do so is to sort features by Group, and look for unused features that are next to the frequently used ones. You can quickly check how to use the feature by selecting it and studying the corresponding tip that opens.

Reporting Issues and Sharing Your Feedback

IntelliJ IDEA provides various means to report problems and seek for assistance. In this section:

- Locating IntelliJ IDEA log
- Configuring IntelliJ IDEA log settings
- Reporting issues
- Sharing feedback
- Seeking assistance

Locating IntelliJ IDEA log

On certain occasions, you will be required to attach the IntelliJ IDEA log to an email to the support service. You can easily locate the log file as described below.

- On the main menu, choose Help | Show Log in Explorer (Windows and Linux), or Help | Show Log in Finder (OS X). The Explorer/Finder opens, with the log file selected.

Configuring IntelliJ IDEA log settings

To avoid editing the `log.xml` file itself, IntelliJ IDEA suggests a handy dialog box to change logging level for a category. This file resides under the `bin` directory of IntelliJ IDEA installation.

1. On the main menu, choose Help | Configure Debug Log Settings.
2. In the dialog box that opens, type the log categories names, separated with new lines.

While editing `log.xml`, keep in mind the following:

- It is not recommended to change `log.xml`, because from time to time it causes problems with patches.
- Editing `log.xml` should be done in tight contact with the support service. The reason is that the users might be unaware of the modules names to be specified, while the support service can suggest modules for the better diagnostics.

Reporting issues

1. Open the IntelliJ IDEA tracking system at https://youtrack.jetbrains.com/ .
2. Click Create issue.
3. On the page that opens, choose IntelliJ IDEA from the Project drop-down list.
4. Describe your problem and provide a brief summary of it in the Description and Summary fields respectively.
5. If necessary, attach a screenshot that illustrates your problem.
6. Click Create issue when ready.

Sharing feedback

To share your feedback, do one of the following:

- Choose Help | Submit Feedback, which redirects you to the online feedback form.
  This form enables you to create a IntelliJ IDEA-specific YouTrack issue.
- Click the Comments link on each page of online version of the IntelliJ IDEA documentation. In the Comments area, vote for the page, and leave your comments.

Seeking assistance

To find assistance, do one of the following:

- Apply to the JetBrains Support
- Write to the support service. Use the following address:
  - intellij-support@jetbrains.com
  - idea-support@jetbrains.com

  If necessary, attach the source code and the IntelliJ IDEA log .
- Ask the IntelliJ IDEA Community .

Keymap Reference

IntelliJ IDEA provides the default platform-specific keymap reference in `pdf` format.

You can view the keymap either from the built-in documentation, or on the web.

– Keymap for Windows/Linux ,Keymap for OS X

How-to

This part provides descriptions of the platform and language-specific procedures.

- General Guidelines
- Language and Framework - Specific Guidelines

General Guidelines

This part provides descriptions of procedures required to fulfil the platform tasks with IntelliJ IDEA.

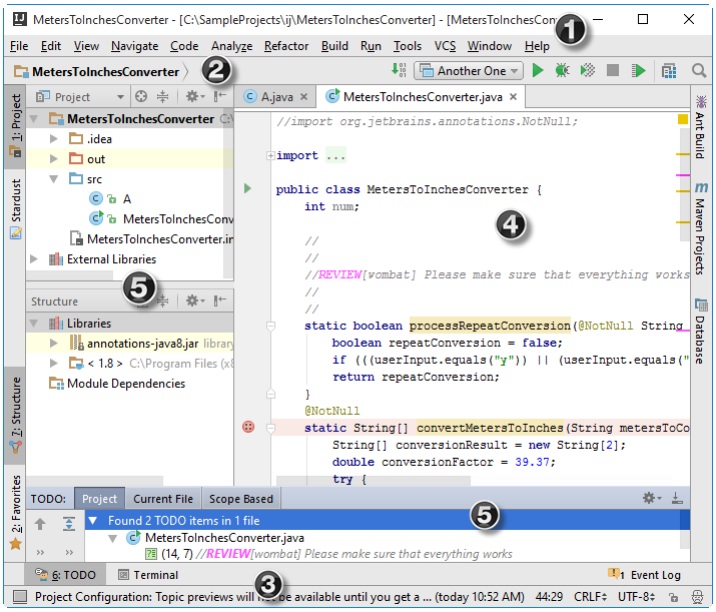
– Guided Tour around the User Interface
– IntelliJ IDEA Editor
– Tool Windows
– Creating and Managing Projects
– Creating and Managing Modules
– Configuring Project and Working Environment
– Configuring Project Structure
– Managing Facets
– File and Code Templates
– Live Templates
– Populating Projects
– Generating Code
– Auto-Completing Code
– Creating and Optimizing Imports
– Analyzing Applications
– Code Inspection
– Intention Actions
– Annotating Source Code
– Documenting Source Code in IntelliJ IDEA
– Viewing Reference Information
– Using Language Injections
– Internationalization and Localization Support
– Navigating Through the Source Code
– Searching Through the Source Code
– Compiling Applications
– Annotation Processors Support
– Working with Run/Debug Configurations
– Running
– Debugging
– Testing
– Working with Artifacts
– Deployment. Working with Web Servers
– Code Coverage
– Refactoring Source Code
– Using Local History
– Version Control with IntelliJ IDEA
– Managing Plugins
– Managing Tasks and Context
– Using Command Line Tools
– Using File Watchers
– Working with Embedded Local Terminal
– Comparing Files and Folders
– Working with Background Tasks
– Working with IntelliJ IDEA Features from Command Line

Guided Tour around the User Interface

This chapter gives you insight into how IntelliJ IDEA user interface is organized to help you find your way through your working environment.

> **Tip** This chapter outlines the default (out-of-the-box) IDE interface layout. Note that plugins and other add-ons you are installing may change the way your IDE looks and behaves, for example there can be extra command buttons or menu items appearing in uncommon locations.

When you first run IntelliJ IDEA, or have no open project, IntelliJ IDEA displays the Welcome screen , which enables quick access to the major entry points. When a project is opened, IntelliJ IDEA displays the main window. This window is made up of the logical areas, which are shown on the picture below, marked with number labels.



1. Menus and toolbars - the main menu and toolbars let you carry out various commands.
2. Navigation bar that helps navigate through the project and open files for editing.
3. The status bar - indicates the status of your project, the entire IDE, and shows various warning and information messages.
4. The editor - here you create and modify the code.
5. IntelliJ IDEA tool windows - secondary windows that provide access to various specific tasks (project management, search, running and debugging, integration with version control systems, etc.).

Welcome Screen

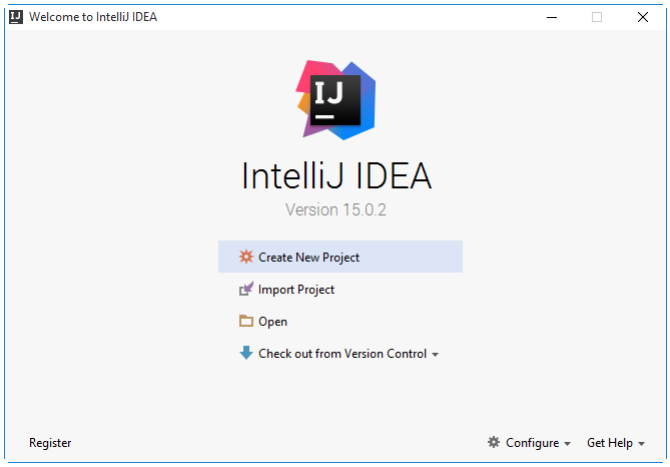
On this page:

- Overview
- Recent Projects
- Quick Start

Overview

IntelliJ IDEA displays the Welcome screen when no project is open. From this screen, you can quickly access the major starting points of IntelliJ IDEA. The Welcome screen appears when you close the current project in the only instance of IntelliJ IDEA. If you are working with multiple projects, usually closing a project results in closing the IntelliJ IDEA window in which it was running, except for the last project, closing this will show the Welcome screen.

The Welcome screen is divided into the following sections: Quick Start and Recent Projects (if any).

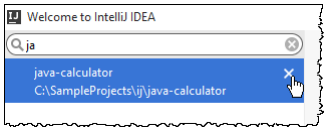


Tip Use the Tab key to navigate through the Welcome screen.

Recent Projects

If appropriate, the left-hand pane shows a list of projects you've recently been working with. Click a project to reopen it.

To find a project of interest, start typing its name.



To delete a recent project from the list, follow these steps:

1. Use the Tab key to move focus into the list of Recent projects.
2. Use arrows keys to select the project you'd like to remove, or find it, as shows above.
3. Do one of the following:
   - Press Delete on your keyboard and confirm deletion in the Remove Recent Project dialog box that opens.
   - To remove the selected recent project silently, click ✕ or choose Remove Selected from Welcome Screen on the context menu of the selection.

Quick Start

Use the links of this section to create a new project , open an existing project, or check out a project from version control.

Also, use the drop-down arrows (▾) Configure to configure your working environment and default project , and Get Help to open help topics, tips of the day, and default keymap document.

Menus and Toolbars

On this page:

– Overview
– Main elements of IntelliJ IDEA window
– Tips and tricks

Overview

IntelliJ IDEA menus and toolbars let you carry out various commands. The main menu and toolbar contain commands that affect the entire project or large portions of it. Additionally, context-sensitive pop-up menus let you perform the commands, which are specific to a part of a project, like source file, class, etc. Almost each of the commands has an associated keyboard shortcut to enable quicker access to it.

Use the check commands of the View menu to show or hide the main elements of the IntelliJ IDEA window. For example, if you want to show the main toolbar, make sure that the check command Toolbar is selected.

Main elements of IntelliJ IDEA window

1. Main menu

   The main menu contains commands for opening, creating projects, refactoring the code, running and debugging applications, keeping files under version control and more.

2. Main toolbar

   The main toolbar contains buttons that duplicate the essential commands for quicker access. You can hide the main toolbar, using the checked command on the toolbar context menu.

   By default, the main toolbar is hidden. To show it, select the check command View | Toolbar on the main menu.

3.Navigation bar

   Navigation bar is a quick alternative to the Project tool window.

   By default, the Navigation bar is shown. To hide it, clear the check command View | Toolbar on the main menu.

4. Context menus

   These menus, available with right-click, contain commands applicable to the current context.

5. Pop-up menus

   These menus, available with `Alt+Insert`, contain commands applicable to the current context.

Tips and tricks

– Show or hide the main elements of IntelliJ IDEA UI using the View menu.
– Descriptions of the actions from all the menus and toolbar buttons are displayed in the left side of the Status bar.
– If you know which action you want to perform, but do not know where to find the appropriate command, just press `Ctrl+Shift+A`, type some part of the name of action you want to perform, and select the desired action from the suggestion list.

Navigation Bar

On this page:

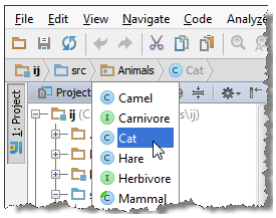
– Introduction

– Toggling the Navigation Bar

– Tips and tricks

Introduction

Navigation Bar is a quick alternative to the Project view . Use this tool to navigate through the project and open files for editing .



Toggling the Navigation Bar

To show the Navigation Bar, do one of the following

– On the View menu, select the check command Navigation Bar.

– Press `Alt+Home` .

To hide the Navigation Bar

– On the View menu, clear the check command Navigation Bar.

Tips and tricks

Please note the following:

– When the main toolbar is hidden, the Navigation Bar shows the run/debug configuration selector, run and debug , project structure , version control buttons (if version control integration is enabled) and search everywhere magnifying glass .

– When the Navigation bar is hidden, press `Alt+Home` to open it floating.

– Pressing `Escape` returns focus to the editor.

Status Bar

On this page:

– Introduction

– Status Bar icons

Introduction

IntelliJ IDEA Status Bar indicates the current IDE state and lets you carry out certain environment maintenance tasks.

Status Bar icons

Icon	Description
	Click to toggle showing or hiding tool window bars. Also double press and hold ⌘ (for OS X) or Alt (for Windows or *NIX) to show hidden tool window bars. Refer to the procedure description .
Load offline inspection results	This section of the Status bar shows description of a command, currently selected on the main menu, context menu, or a toolbar.
	Click this icon to invoke the Background Tasks manager . Visibility of this icon in the Status bar depends on a launched background task.
48:82 48:82/17	The first two numbers denote the current caret position in the editor(line and column). If a selection is made in the editor, IntelliJ IDEA shows the length of the current selection after slash.
	This Lock icon indicates the read-only or writable attribute of the current file in the editor. To toggle the file attribute, click the Lock icon or use the File | Make File Writable/Read-only command on the main menu.
CRLF	View and change line endings of the current file in the editor.
windows-1251	View and change encoding of the current file in the editor.
	Click this icon to navigate to the pending source control changelists in the Incoming tab of the Version Control tool window.
	Hovering your mouse pointer over the icon shows the current code inspection profile at the tooltip. Clicking the Hector icon results in showing a dialog box with the following functions: – Highlighting level. Use the slider to change highlighting level for the current file, or configure inspection profile . Depending on the highlighting level selected by the slider, Hector keeps an eye on the code (Inspection level), turns half face (Syntax), or averts his face from the code (None). – Power Save Mode. Select this check box to minimize power consumption of your computer on account of eliminating the background operations. To indicate that the mode is on, Hector fades . When Power Save Mode is on, IntelliJ IDEA reduces its functionality to the one of a text editor, by not executing expensive background activities that drain laptop battery. These activities include error highlighting and on-the-fly inspections, autopopup code completion , and automatic incremental background compilation . You can also toggle Power Save Mode through the File | Power Save Mode command on the main menu. – Import popup. Use this check box to enable or disable auto-import for the current file. You can turn auto-import off for the whole product in the Auto Import page of the Settings/Preferences dialog.
	Indicates that there are unattended notifications. Click this icon to see the notification descriptions in the Event Log tool window.
	Alternatively, when the icon is empty, there are no new notifications.
	This blinking icon indicates that internal IDE errors have occurred. Click to view the error descriptions and submit reports.
118M of 494M	Shows the current heap level and memory usage. Visibility of this section in the Status bar is defined by the Show memory indicator check box in the Appearance page of the Settings/Preferences dialog. It is not shown by default. Click the memory indicator to run the garbage collector.
	Shows logged in to or logged out from IntelliJ Configuration Server Server status. Click to show the IntelliJ Configuration Server Settings dialog box. This is only available upon installing the IntelliJ Configuration Server plugin.

Note More icons appear in the Status Bar as you download and install plugins.

IntelliJ IDEA Viewing Modes

On this page:

- Basics
- Toggling full screen mode
- Toggling presentation mode
- Toggling distraction-free mode

Basics

IntelliJ IDEA provides special view modes:

- Full Screen mode allows you to use the entire screen for coding. This removes all menus from view, as well as the operation system controls. However, you can use context menus and keyboard shortcuts. The main menu is also available when you hover the mouse pointer over the top of the screen.
- Presentation mode - this mode is similar to the Full Screen mode, but it is designed for making presentations that involve coding with IntelliJ IDEA. In this mode, IntelliJ IDEA increases the font size and hides everything except the editor. If necessary, tool windows can be also displayed in this view using the corresponding items in the View | Tool Windows menu.
- Distraction-free mode shows no toolbars, no tool windows, no editor tabs; the code is center-aligned; etc.

Toggling full screen mode

Besides manipulating the tool windows (show/hide or resize them), IntelliJ IDEA makes it possible to maximize the entire product window, hiding the main menu.

To enter full screen mode

- On the main menu, choose View | Enter Full Screen.

To exit full screen mode

1. Hover your mouse pointer over the top-most line to reveal the main menu.
2. On the main menu, choose View | Exit Full Screen.

Toggling presentation mode

In the presentation mode, the editor occupies the entire screen, while all the other IntelliJ IDEA components are hidden.

Besides that, the font size in this mode is larger than usual. You can define font size for the presentation mode in the Appearance page of the Settings dialog.

To enter presentation mode

- On the main menu, choose View | Enter Presentation Mode.

To exit presentation mode

1. Hover your mouse pointer over the top-most line of the editor to reveal the main menu.
2. On the main menu, choose View | Exit Presentation Mode.

Toggling distraction-free mode

In the distraction-free mode, the editor occupies the entire IntelliJ IDEA frame, without any editor tabs and tool-window buttons. The code is center-aligned.

To enter distraction-free mode

- On the main menu, choose View | Enter Distraction-Free Mode.

To exit distraction-free mode

- On the main menu, choose View | Exit Distraction-Free Mode.

Tip These actions are not mapped to any shortcuts by default. You can create your own shortcuts as described in the section Configuring keyboard shortcuts .

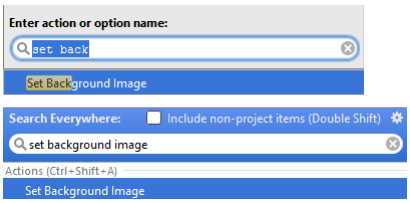
Setting Background Image

IntelliJ IDEA allows you to define any image as a background. So doing, you can set a background image for the current project only, or for any project you open or create anew.

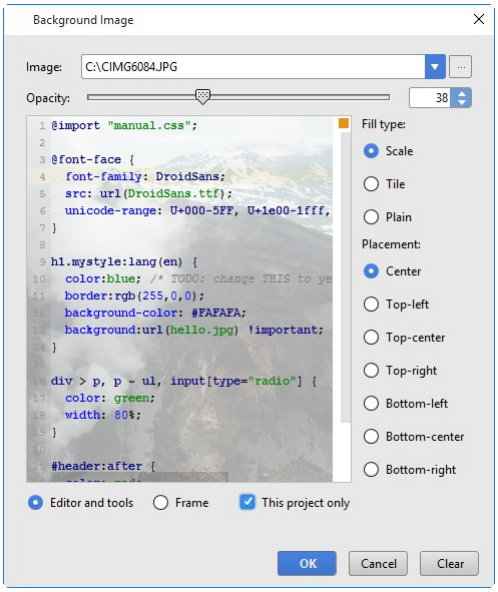
This feature has no keyboard shortcut (you can easily create a shortcut as described in the section Configuring Keyboard Shortcuts).

To set a background image

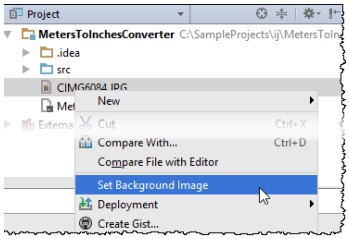
1. Do one of the following:
   - Press `Shift` twice (see Searching Everywhere).
   - Press `Ctrl+Shift+A` (see Navigating to Action).



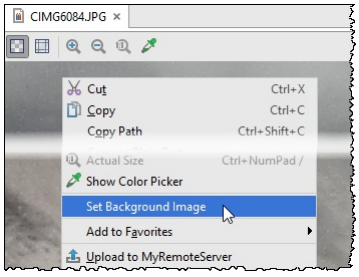
2. In the dialog box that opens, specify the image you want to use as the background, its opacity, filling and placement options. Besides that, you can choose to show background in the editor and tool windows, or in the IntelliJ IDEA frame.
   Also, select the check box This project only to show background in the current project, and ignore this background in the other projects.



If an image is already selected in a IntelliJ IDEA project, this action is also available from the context menu of the Project tool window:



and in the image editor



Editor

In this section:

- Editor
  - Basics
  - Active editor
  - Editor's areas
- Basic Editing Procedures
- Advanced Editing Procedures
- Managing Editor Tabs
- Using TODO

Basics

IntelliJ IDEA editor is a powerful tool for creating and modifying source code. As any other IDE editor, it supports basic features like bookmarks , breakpoints , syntax highlighting , code completion , zooming , folding code blocks , etc. There are, however, plenty of advanced features like macros , highlighted TODO items , code analysis , intention actions , intelligent and fast navigation , and a lot more.

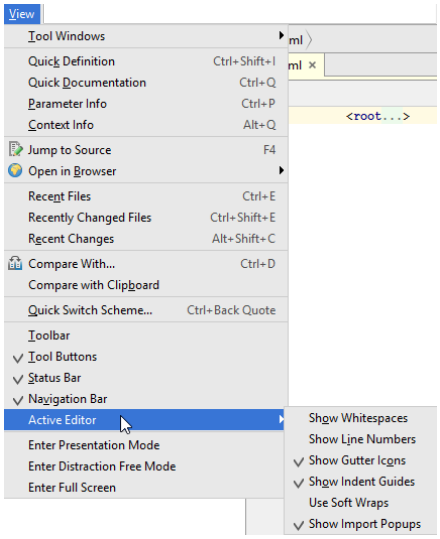
To configure your editing environment, use the Editor settings page and its child pages. There is also a Quick Switch Scheme command that lets you change color schemes, themes, keymaps, etc. with a couple of keystrokes.

The editor is tab-based. All operations with the editor tabs are available from the context menu of a tab , or from Window | Editor tabs node of the main menu .

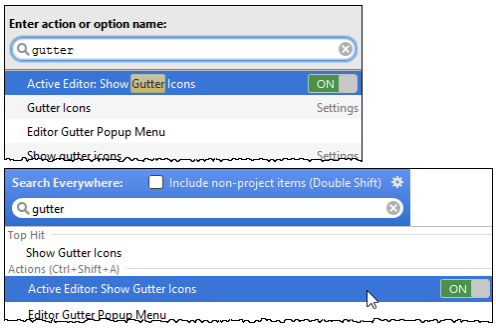
Active editor

Tip You always return the focus to the active editor from any tool window by pressing the Escape key.

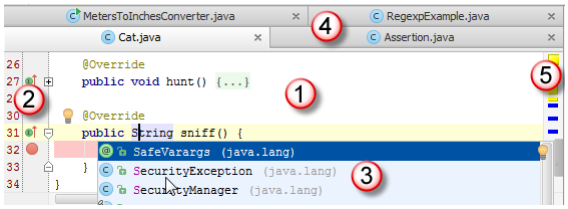
When you open a file for editing , it opens in its own tab. The editor you are currently working in, is the active editor. You can change behavior of the active editor using the commands under View | Active Editor node of the main menu:



Alternatively, you can invoke the commands related to the active editor, from Find Action or Search Everywhere :



Editor's areas



1. Editor area

   Use this area to type and edit your source code. The editor suggests numerous coding assistance facilities. Refer to the sections under this node, and to Basic Editing

Procedures and Advanced Editing Procedures for details.

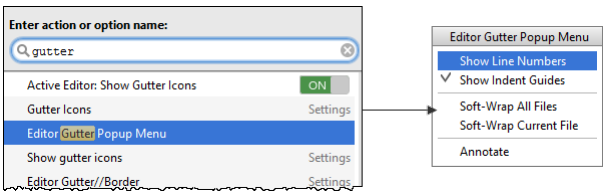
2. Gutter area

   The left gutter provides additional information about your code and displays the various icons that identify the code structure, bookmarks, breakpoints, scope indicators, change markers and the code folding lines that let you hide arbitrary code blocks.

   You can change the behavior of the left gutter.

   For example, it's possible to make the left gutter thinner by hiding the gutter icons. This is done either for the active editor , or for all the newly created editors .

   To change the behavior of the left gutters, use either the Appearance page of the editor settings, or the Editor Gutter Popup Menu:



   By default, this command is not mapped to any keyboard shortcut. You can create your own shortcut as described in the section Configuring Keyboard Shortcuts .

3. Smart completion pop-up

   This is one of the key editing assistance features that suggests method names, functions, tags and other keywords you are typing.

4. Document tabs

   Enable quick navigation across the multiple documents you are working on. Clicking a tab brings its contents to front and makes it available for editing in the active editor.

   To move between the tabs, use the keyboard shortcuts `Alt+Right` or `Alt+Left`.

   Clicking a tab while the `Ctrl` / ⌘ key is pressed, allows navigating to any part of the file path, through opening it in an external browser.

   Context menu of a tab provides all commands applicable to a file opened in the editor, for example:

   - Close one or more tabs.
   - Pin active tab.
   - Split and unsplit tabs.
   - Manage groups of tabs.
   - Navigate between tabs.
   - Add to Favorites.
   - Move to a changelist.
   - Run , or debug in the active editor.
   - Perform local history and version control commands.
   - Perform commands of your own tools .

   By default, the tabs appear on top of the editor, but you can change their location as described in the section Changing Placement of the Editor Tab Headers .

5. Validation side bar / marker bar

   This is the bar to the right from the editing area, showing the green, red or yellow box on its top depending on whether your code is okay, or contains errors or warnings. This bar also displays active red, yellow, white, green and blue navigation stripes that let you jump exactly to the erroneous code, changed lines, search results, or TODO items.

Basic Editing Procedures

This section describes how to perform the most common editing tasks:

- Selecting Text in the Editor
- Cutting, Copying and Pasting
- Copy and Paste Between IntelliJ IDEA and Explorer/Finder
- Commenting and Uncommenting Blocks of Code
- Undoing and Redoing Changes
- Opening and Reopening Files in the Editor
- Closing Files in the Editor
- 'Lens' Mode
- Multicursor
- Adding Editors to Favorites
- Saving and Reverting Changes
- Zooming in the Editor

Selecting Text in the Editor

On this page:

- Basics
- Selecting all text in the active editor tab
- Selecting with navigation keys
- Extending selection
- Shrinking selection
- Multiselection
- Toggling between line and column selection modes
- Using column selection mode
- Using smart selection
- Sticky selection
- Tips and tricks

Basics

The basic way to select a piece of text is to extend the selection with the mouse cursor. IntelliJ IDEA, as a keyboard-centric IDE, suggests to use navigation keys to make selections.

You can opt to select pieces of text by lines, or select rectangular fragments in the column mode, extend and shrink the selection, use multiselection, and sticky selection.

Selecting all text in the active editor tab

To select the entire text in the current editor tab, do one of the following:

- On the main menu, choose Edit | Select All.
- Press `Ctrl+A`.

Selecting with navigation keys

To select text from the current caret position to the beginning/end of the current word:

- `Ctrl+Shift+Left`, `Ctrl+Shift+Right`.

To select text from the caret position to the beginning/end of the current line:

- Double-click `Ctrl` and press `Home`/ `End`

To select text from the current caret position to the top/bottom of the screen:

- `Ctrl+Shift+Page Up`, `Ctrl+Shift+Page Down`.

Extending selection

To extend selection from the word at caret to the piece of code the caret is contained in, do one of the following:

- On the main menu, choose Edit | Extend Selection
- Press `Ctrl+W` to select the word where the caret is currently located.
- Press `Ctrl+W` successively to extend selection to the next containing node (for example, an expression, a paired tag, an entire conditional block, a method body, a class, a group of vararg arguments, etc.)
  While extending selection, keep in mind that:
  - Pressing `Ctrl+W` successively in plain text or comments extends the selection first to the current sentence, then to the current paragraph.
  - Pressing `Ctrl+W` successively in a method call that contains vararg arguments, first selects a vararg argument the caret rests at, then the whole group of vararg arguments, and then all arguments in the method call.

Shrinking selection

To shrink selection in the reverse order (from the outermost container to the word where the caret currently resides), do one of the following:

- On the main menu, choose Edit | Shrink Selection
- Press `Ctrl+Shift+W`.

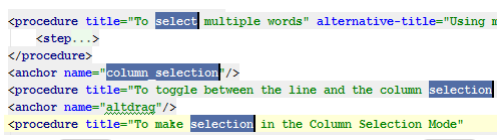
> **Tip** The selection extends or shrinks according to capitalization, if the Use "CamelHumps" words check box is selected on the Smart Keys page of the editor settings.
>
> If you want to make selection according to capitalization, using double-click, make sure that the check box Honor CamelHumps words... is selected on the General page of the editor settings.

Multiselection

IntelliJ IDEA supports selecting multiple text fragments. So doing, one can select multiple words, lines or rectangles.

To select multiple words, follow these steps

1. Do one of the following:
   - Press `Shift+Alt` and double-click the left mouse button.

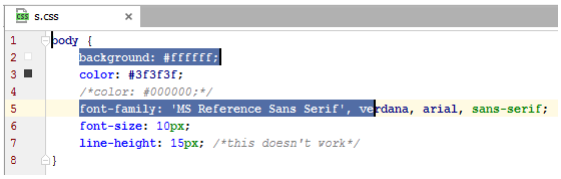


   - Press `Alt+J`, or select some text fragment. Then press `Alt+J` again, to find the matching piece of text.

2. After selection is complete, you can start editing all the fragments as if they were one.

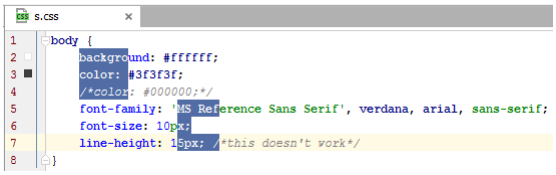
To select multiple fragments of text, follow these steps

1. Press `Shift+Alt`
2. Drag the mouse pointer



To select multiple rectangular fragments of text, follow these steps

1. Press `Ctrl+Shift+Alt` (Windows or UNIX)/ `Shift+Alt+Cmd` (OS X)
2. Drag the mouse pointer



Refer to the section Multicursor for additional information.

Toggling between line and column selection modes

To toggle between the line and the column selection modes, do one of the following:

– On the main menu, choose Edit | Column Selection Mode.

– On the context menu of the editor, choose Column Selection Mode.

– Press `Shift+Alt+Insert`.

Using column selection mode

To make selection in the Column Selection Mode, do one of the following:

– Keeping the `Alt` key pressed, drag your mouse pointer to select the desired area.

– Keeping the middle mouse button pressed, drag your mouse pointer to select the desired area.

– Press `Shift+Alt` and the middle mouse button. This is specially helpful, if you want to avoid dragging.

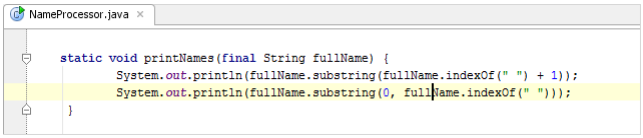
Using smart selection

When you perform various code refactorings that involve selecting an expression IntelliJ IDEA can help you select the expression of interest. This feature is known as smart expression selection .

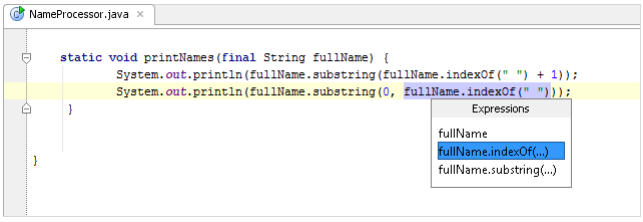
The chapter for the Extract Variable refactoring is just an example. The smart expression selection feature, in fact, is available in all the refactorings that start with selecting an expression.

To use smart expression selection, follow these steps:

1. Place the cursor before or within the expression.



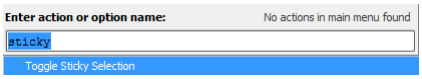
2. Choose Refactor | Extract Variable on the main menu or on the context menu. or press `Ctrl+Alt+V`.
3. In the Expressions pop-up menu, select the expression. To do that, click the required expression. Alternatively, use the Up and Down arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



Sticky selection

To toggle sticky selection, do one of the following:

– Press `Ctrl+Shift+A`, in the pop-up frame type sticky, and choose Toggle Sticky Selection from the suggestion list:



- In the Emacs keymap, use keyboard shortcut `Ctrl+Space`.

Tips and tricks

- When sticky selection is on, you can turn it off by invoking copy or cut, or by toggling it again.
- To create a large rectangular selection, create a normal selection first, with the given starting and ending points, and then press `Shift+Alt+Insert` to toggle to the column selection mode.

Cutting, Copying and Pasting

On this page:

- Basics
- Copying a Selected Fragment of Text
- Copying the path to a file
- Copying the reference to a line or a symbol
- Cutting a selected fragment of text
- Pasting the last entry from the clipboard
- Pasting the last entry from the clipboard as plain text
- Pasting a specific entry from the clipboard

Basics

IntelliJ IDEA provides a number of handy Clipboard operations. You can copy, cut, and paste selected text, a path to a file, or a reference to a symbol or a line of code.

Because IntelliJ IDEA uses the system Clipboard, you can copy and paste between applications. So doing, when pasting Clipboard entries, IntelliJ IDEA removes any formatting from the text and any special symbols from the `String` values.

The Paste command smartly understands what is being inserted. If you paste a reference to a symbol, it is analyzed for possible imports, references, etc. So doing, IntelliJ IDEA provides the necessary brackets and places the caret at the appropriate insertion point. The Paste Simple command helps paste any Clipboard entry as a plain text, without any analysis.

IntelliJ IDEA enables Clipboard stacking, which means that you can store multiple Clipboard entries and access them with a single shortcut. The number of entries that can be kept in the Clipboard stack is customizable on the Editor page of the Settings/Preferences dialog.

Copying a Selected Fragment of Text

Do one of the following:

- On the main menu, choose Edit | Copy.
- Press `Ctrl+C`.
- Click the Copy button on the toolbar.

Note that the `Ctrl+D` keyboard shortcut clones a line at the caret or a selected arbitrary fragment of text.

Copying the path to a file

Do one of the following:

- Open the desired file in the editor, then choose Edit | Copy Path on the main menu or press `Ctrl+Shift+C`.
- Select the desired file in the Project tool window and choose Copy Path on the context menu of the selection.

Copying the reference to a line or a symbol

1. Open the desired file in the editor.
2. Place the caret at a certain line of code.
3. Do one of the following:
   - On the main menu, choose Edit | Copy Reference
   - On the context menu of the line at caret, choose Copy Reference
   - Press `Ctrl+Shift+Alt+C`.

   IntelliJ IDEA creates a string in the format that depends on a symbol at caret. For example:

   `/MetersToInchesConverter.java:14` for a Java class (format `<fully qualified path>:<line number>`)

   `java.io.PrintStream#println(java.lang.String)` for a Java method (format `<full class name>.<method_signature>`)

Cutting a selected fragment of text

1. Select the desired fragment in the editor.
2. Do one of the following:
   - On the main menu, choose Edit | Cut.
   - Press `Ctrl+X`.
   - Click the Cut button on the toolbar.

Pasting the last entry from the clipboard

1. Place the caret in the location where you want to paste content.
2. Do one of the following:
   - On the main menu, choose Edit | Paste.
   - Press `Ctrl+V`.
   - Click the Paste button on the toolbar.

   If you perform paste in a Java file, the further behavior depends on the settings in the Auto Import page of the Editor options. If the Ask option has been selected, select the necessary imports from the list of missing imports. In all other cases, the last clipboard entry is pasted silently.

Pasting the last entry from the clipboard as plain text

Do one of the following:

- On the main menu, choose Edit | Paste Simple

– On the main menu, choose Edit | Paste Simple.
– Press `Ctrl+Shift+Alt+V`.

Pasting a specific entry from the clipboard

1. On the main menu, choose Edit | Paste from History or press `Ctrl+Shift+V`.
2. In the Choose Content to Paste dialog box select the desired entry from the list of recent Clipboard entries, and click OK.

The depth of the Clipboard stack is configured in the Limits section on the Editor page of the Settings dialog box. When the specified number is exceeded, the oldest entry is removed from the list.

Copy and Paste Between IntelliJ IDEA and Explorer/Finder

IntelliJ IDEA enables tight interaction with the native file managers (Explorer on Windows, or Finder on Mac), and allows you exchange files and directories via the system clipboard, using numerous techniques:

- Cut, copy and paste files and directories from the Project tool window of IntelliJ IDEA to a directory in the file manager, and vice versa, using menu commands and keyboard shortcuts:

Keyboard shortcut	Function	Use this shortcut to...
Ctrl+C	Copy	Copy selected text to the Clipboard.

- Move (drag) or copy (Ctrl +drag) a file or directory from the file manager to a directory in the Project tool window.
- Move (drag) or copy (Ctrl +drag) a file from the Project tool window to a directory in the file manager.
- Open any file for editing, by dragging it from a file manager to the editor.

Commenting and Uncommenting Blocks of Code

On this page:

– Basics
– Configuring commenting behavior
– Commenting and uncommenting lines of code
– Commenting and uncommenting blocks of code

Basics

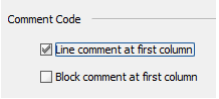
You can comment or uncomment the current line or selected block of source code.

Commenting feature extends to all supported file types. For the custom file types, you can define line and block comments characters, as described in the section Creating and Registering File Types .

Configuring commenting behavior

It is possible to force IntelliJ IDEA to place Java comments strictly at the left margin. To do that, in the Settings dialog, open Code Style | Java , and click Code Generation tab .

In the section Comment Code select the required check boxes:



If the check boxes are cleared, the comments characters are placed at the current caret position.

Commenting and uncommenting lines of code

Do one of the following:

– On the main menu, choose Code | Comment with Line Comment.
– Press `Ctrl+Slash`.

```
//      {
//          user = em.merge(user);
//          facesMessages.add("Password updated");
//          changed = true;
//      }
```

Commenting and uncommenting blocks of code

To add or remove a block comment, do one of the following:

– On the main menu, choose Code | Comment with Block Comment.
– Press `Ctrl+Shift+Slash`.

```
/*  {
        user = em.merge(user);
        facesMessages.add("Password updated");
        changed = true;
    }*/
```

Undoing and Redoing Changes

On this page:

– Basics
– How it works?
– Undoing and redoing changes

Basics

The Undo command discards the last changes to the file in the editor. The Redo command discards the results of the last Undo command.

You can undo or redo your changes as many times as required. However, when you exit IntelliJ IDEA, the undo history is lost.

IntelliJ IDEA smartly defines the logical steps that can be undone and redone. The following events signal about the end of a logical step:

– Pressing `Enter`.
– Repositioning the mouse cursor.
– Using navigation keyboard shortcuts.
– Cutting or pasting.
– Pressing `Tab`.

IntelliJ IDEA expands the undo and redo mechanism to complex operations, such as reformatting or refactoring source code, creating or deleting files. When you undo or redo a complex operation, IntelliJ IDEA requests for your confirmation.

How it works?

IntelliJ IDEA moves the caret before each step of undo/redo, and then performs the Undo/Redo actions.

Undoing and redoing changes

To undo an action, do one of the following

– On the main menu, choose Edit | Undo.

– Press `Ctrl+Z`.

To redo an action, do one of the following

– On the main menu, choose Edit | Redo.

– Press `Ctrl+Shift+Z`.

Opening and Reopening Files in the Editor

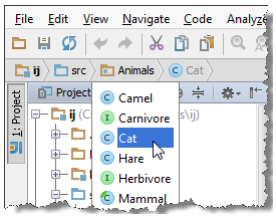
On this page:

- Opening files for editing
- Opening external files
- Reopening files
- Opening files in a separate window

Opening files for editing

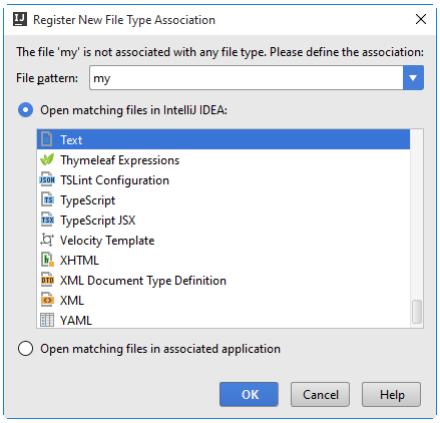
To open a file for editing

1. Do one of the following:
   - Double-click the desired file in one of the Tool Windows .
   - Select the desired file in one of the Tool Windows and press `F4` .
   - Select the desired file in the one of the Tool Windows and choose Jump to Source on the context menu.
   - Use the Navigate command for a Class, File, or Symbol.
   - Click the desired directory in the Navigation bar , and select file from the drop-down list:



2. If the file type is registered, the file opens silently in IntelliJ IDEA's editor.
   If the file type is registered under the category Files opened in associated applications, it will be opened in its associated application, rather than in the IntelliJ IDEA editor. By default, IntelliJ IDEA suggests a number of such file types, for example `.doc` , `.chm` , or `.pdf` .

   If the file type is unknown, IntelliJ IDEA suggests you to choose whether you want to register a new file type, or open such file in its associated application. Specify your choice in the Register New File Type Association dialog box:



   You can register the required file types on the File Types page of the Settings/Preferences dialog.

The maximum size of files parsed by IntelliJ IDEA is controlled by the `idea.max.intellisense.filesize` setting in `idea.properties` file.

The file `idea.properties` , located in the `bin` directory of the IntelliJ IDEA installation folder, should not be edited. Instead of editing the original `idea.properties` , create file `idea.properties` in the location specified below, open it for editing and add the required properties.

So, depending on your platform:

- For Windows: in `%USERPROFILE%\.IntelliJIdeaXX` or `%USERPROFILE%\.IdeaICXX`
- For *NIX: in `~/.IntelliJIdeaXX` or `~/.IdeaICXX`
- For OS X: in `~/Library/Preferences/IntelliJIdeaXX` or `~/Library/Preferences/IdeaICXX`

Note that the larger the file is, the slower its editor works and the higher overall system memory requirements are.

Opening external files

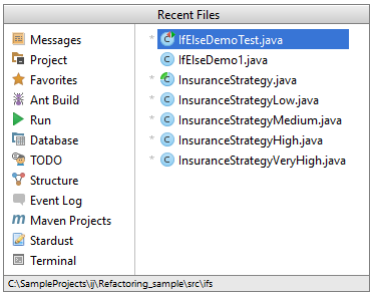
Do one of the following:

- Choose File | Open on the main menu and select the desired file in the dialog box that opens.
- Drag the required file from the Explorer (Windows), File Browser (Linux), or Finder and drop it to the editor. The file opens for editing in a new tab.

Reopening files

To reopen a file

- To open a recently opened file, choose View | Recent Files on the main menu or press `Ctrl+E` . Then select the desired file from the Recent Files pop-up window, that opens.



- To open a recently updated file, on the main menu, choose View | Recently Changed Files or press `Ctrl+Shift+E`. Then select the desired file from the Recently Edited Files pop-up window, that opens.

  Tip Use Recent files limit text box in the Editor settings page to define the maximum number of recent files.

Opening files in a separate window

To open a file in a separate IntelliJ IDEA window

Do one of the following:

- Drag and drop an editor tab outside of the current IntelliJ IDEA window.
- Press `Shift+F4` for a file selected in the Project tool window.
- `Shift+mouse double click` on a file name in the Project tool window.

Closing Files in the Editor

IntelliJ IDEA suggests several ways to close editor tabs.

To close a file in the editor, do one of the following

- On the main menu point to Window | Editor Tabs, choose one of the appropriate closing commands.

  Close

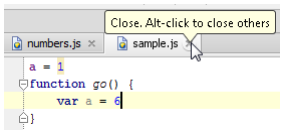
  Closes the file in the active tab.

  Close All

  Closes all editor tabs.

  Close Others

  Closes all tabs except the current one. The alternative way to close all other tabs lays with clicking the × button, while holding the `Alt` key pressed:



  Close Unmodified

  Closes all files that were not changed. This command is only available, when version control integration is enabled in project.

  Close All But Pinned

  Closes all files that were not pinned. This command appears, if there are pinned editor tabs.

- Right-click any editor tab, and choose same commands on the context menu.

- Point with your mouse cursor to a tab and click the middle mouse button.

- Point with your mouse cursor to a tab and click ×.

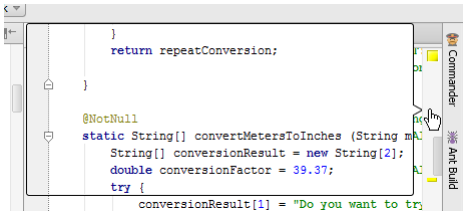
- Press `Ctrl+F4`.

Tip When you close modified files, IntelliJ IDEA preserves all changes in the current editing session. After reopening such files, the results of editing are restored.

'Lens' Mode

Hover the mouse pointer over a warning, error stripe or just some section on the scroll bar outside of the scroll box. IntelliJ IDEA shows the source code fragment annotated with the warning/error messages.

This helps viewing the context a marker applies to, and the source code outside of the editor visible area.



This behavior is enabled by default.

To toggle the lens mode

1. Do one of the following:
   - Open the Appearance page of the Settings dialog.
   - Right-click the code analysis marker on top of the current editor.
2. Select or clear the check box Show editor preview tooltip.

Multicursor

On this page:

– Basics

– Adding, deleting, and cloning carets

– Copying and pasting

Basics

IntelliJ IDEA supports multiple carets. The majority of the editor actions, such as keyboard navigation, text insertion and deletion, etc., apply to each caret. Live templates and autocompletion are supported as well.

It is possible to add or delete carets; at least one caret always exists in an editor tab.

The most recently added caret is considered primary. Highlighting of an editor line at caret applies to the primary caret only.

Placement and behavior of the carets depend on:

– Enabled or disabled column selection mode .

– Enabled or disabled placement of caret after the end of line .

Refer to the section Selecting Text in the Editor for additional information.

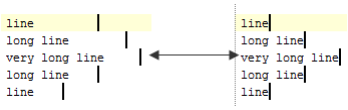
Adding, deleting, and cloning carets

To add carets, do one of the following

– Press `Shift+Alt` and click the left mouse button at the location of the caret.

– Double-click `Ctrl` (Windows or UNIX)/ `Alt` (OS X) and press up or down arrow keys.

The new carets are added to the specified locations, according to setting of the Allow placement of caret after end of line check box:



To delete carets, do one of the following

– Press `Esc` to delete all the existing carets, except the primary one.

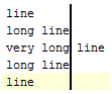
– Press `Shift+Alt` and click the left mouse button on a caret to be deleted.

To clone an existing caret upward or downward, do one of the following:

– Press `Ctrl+Shift+A` , type Clone caret, and choose the desired action from the suggestion list:

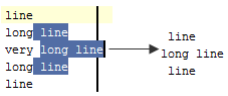
Note that by default these actions are not bound to the keyboard shortcuts. You can do it yourself, as described in the section Configuring keyboard shortcuts .

The primary caret is propagated upwards or downwards:



Copying and pasting

When a text with multiple cursors is copied (`Ctrl+C`) or cut (`Ctrl+X`), selections for each caret are placed to the clipboard. On paste (`Ctrl+V`), text from the clipboard is split into lines.



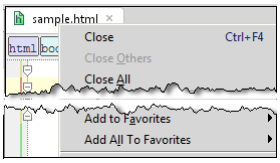
Adding Editors to Favorites

You can group most needed items into Favorite lists and get quick access to them through the Favorites tool window.

To add one or more items to Favorites

1. Do one of the following:
   - Open the desired files in the Editor.
   - Select one or more items in the Project tool window.

2. Right-click the editor tab or the selection in the Project tool window, and choose Add to Favorites on the context menu.



3. On the submenu, specify the Favorites list to add the selected items to. Do one of the following:
   - To add the items to an existing list, select the desired list in the submenu.
   - To create a new list, choose Add to New Favorites List. In the Add New Favorites List dialog box that opens enter the desired group name or accept default settings.

Saving and Reverting Changes

When working with IntelliJ IDEA, you don't need to worry about saving changed files: all changes are auto saved.

Unwanted changes can be undone at any stage of your development workflow. Any file or directory can be reverted to any of the previous states.

– When does IntelliJ IDEA auto save changed files?
– Tuning the autosave behavior
– Using the Save All command
– Marking files with unsaved changes in the editor
– Saving a file under a different name
– Reverting changes

When does IntelliJ IDEA auto save changed files?

Autosave is initiated by:

– Compiling a project, a module or a class
– Starting a run/debug configuration
– Performing a version control operation such as pull, commit, push, etc.
– Closing a file in the editor
– Closing a project
– Quitting the IDE

In fact, there is a lot more autosave triggers, and only the most important ones are mentioned above.

Tuning the autosave behavior

The following options are available for tuning the autosave behavior (File | Settings | Appearance and Behavior | System Settings):

– Save files on frame deactivation (i.e. on switching from IntelliJ IDEA to a different application)
– Save files automatically if application is idle for N seconds

Note that those are optional autosave triggers, and you cannot turn off autosave completely.

Using the Save All command

If necessary, you can initiate saving all changed files yourself. There is the Save All command for that:

– File | Save All
– `Ctrl+S`

Marking files with unsaved changes in the editor

Changed but yet unsaved files can be marked. For this purpose, there is the Mark modified tabs with asterisk option (File | Settings | Editor | General | Editor Tabs).

When this option is on, the files with unsaved changes have an asterisk * on their editor tabs.

Saving a file under a different name

There is no File | Save As command in IntelliJ IDEA. To save a file under a different name or in a different directory, use Refactor | Copy or `F5`.

Reverting changes

You can undo changes by using Edit | Undo or `Ctrl+Z`. To revert files to their previous states, use Local History and corresponding version control functionality .

Zooming in the Editor

IntelliJ IDEA makes it possible to change font size (zoom) in the active editor, and reset font size to the default value. These operations apply to the active editor only. In the other editor tabs, font size is not affected.

On this page:

– Enabling zooming
– Changing font size with the mouse wheel
– Changing font size with the keyboard
– Restoring default font size

To enable changing font size in the editor

1. Open the Settings/Preferences dialog, expand the Editor node, and click General .
2. Make sure that the setting Change font size (Zoom) with Ctrl+MouseWheel is enabled.

To change font size using the mouse wheel

1. Place the caret in the editor.
2. While keeping the `Ctrl` / `⌘` key pressed, rotate the mouse wheel. As you rotate the mouse wheel forward, font size grows larger; as you rotate the mouse wheel backwards, font size decreases.
   The OS X users can use trackpad "Pinch-to-Zoom" gesture to change size of the font and the whole editing area.

To change font size using the keyboard

1. Press `Ctrl+Shift+A` .
2. In the popup frame, type Increase font size or Decrease font size, and then click `Enter` .
   Font grows larger or smaller.

Tip There are no default keyboard shortcuts associated with the actions Increase font size and Decrease font size action. However, you can create your own shortcuts as described in the section Configuring Keyboard Shortcuts .

To reset font size

1. Press `Ctrl+Shift+A` .
2. In the popup frame, type Reset font size, and click `Enter` .
   The default font size is restored.

Tip There is no default keyboard shortcut associated with Reset font size action. However, you can create your own shortcut as described in the section Configuring Keyboard Shortcuts .

Advanced Editing Procedures

This part describes more sophisticated editing techniques provided by IntelliJ IDEA:

- Reformatting Source Code
- Changing Indentation
- Code Folding
- Viewing Current Caret Location
- Toggling Writable Status
- Toggling Case
- Highlighting Braces
- Spellchecking
- Scratches
- Adding, Deleting and Moving Code Elements
- Joining Lines and Literals
- Splitting Lines with String Literals
- Splitting String Literals on Newline Symbols
- Editing CSV and Similar Files in Table Format
- Using Drag-and-Drop in the Editor
- Using Macros in the Editor

Reformatting Source Code

On this page:

- Basics
- Reformatting code
- Reformatting code for the current file
- Skipping a region when reformatting source code
- Example of using formatting markers

Basics

IntelliJ IDEA lets you reformat source code to meet the requirements of your code style. IntelliJ IDEA will lay out spacing, indents, keywords etc. Reformatting can apply to the selected text, entire file, or entire project.

It is also possible to apply reformatting to the parts of the source code only, using the formatting markers .

Reformatting code

To reformat code for a directory, follow these steps:

1. In the Project tool window, select a module or directory you want to apply your reformatting to.
2. From the main menu, select Code | Reformat Code or press `Ctrl+Alt+L` .
   Alternatively, in the Project tool window , right-click the directory and from the context menu, select Reformat Code.
3. In the Reformat Code dialog box, specify the necessary options and filters for you reformatting and click Run.

Reformatting code for the current file

To reformat code for the current file, follow these steps:

1. In the editor of the currently opened file, press `Ctrl+Shift+Alt+L` .
   Note that if you select Code | Reformat Code from the main menu or press `Ctrl+Alt+L` , IntelliJ IDEA will try to reformat the source code automatically without opening the Reformat File dialog.
2. In the Reformat File dialog, specify options for the reformatting and click Run.

Skipping a region when reformatting source code

To enable formatter markers, make sure to select the check box Enable formatter markers in comments in the Code Style page of the Settings/Preferences dialog, and type the markers in the Formatter off/on fields.

To skip a certain region on reformatting, follow these steps:

1. At the beginning of the region, create a line comment (`Ctrl+Slash`), and then manually type the marker specified in the Formatter off field of Code Style page.
2. At the end of the region, create a line comment (`Ctrl+Slash`), and then manually type the marker specified in the Formatter on field of Code Style page.
3. Perform code reformatting, as described above.

Alternatively, create a live template to surround a block of code with formatter off/on markers, see Creating and Editing Live Templates .

Example of using formatting markers

The original source code

```
//@formatter:off
    "scripts": {
        "post-install-cmd": [
            "php artisan optimize"
        ],
        "post-update-cmd": [
            "php artisan clear-compiled",
            "php artisan optimize"
        ],
        "post-create-project-cmd": [
            "php artisan key:generate"
        ]
    },
//@formatter:on
```

The code after reformatting

When the formatting markers are disabled, the original formatting is broken:

```
//@formatter:off
  "scripts": {
      "post-install-cmd": [
          "php artisan optimize"
      ],
      "post-update-cmd": [
          "php artisan clear-compiled",
          "php artisan optimize"
      ],
      "post-create-project-cmd": [
          "php artisan key:generate"
      ]
  },
//@formatter:on
```

When the formatting markers are enabled, the original formatting is preserved:

```
//@formatter:off
    "scripts": {
        "post-install-cmd": [
            "php artisan optimize"
        ],
        "post-update-cmd": [
            "php artisan clear-compiled",
            "php artisan optimize"
        ],
        "post-create-project-cmd": [
            "php artisan key:generate"
        ]
    },
//@formatter:on
```

Changing Indentation

IntelliJ IDEA makes it possible to:

- Indent or unindent text. This action applies to a selection, or to a line at caret.
- Fix wrong indentation according to the code style.
- Choose tabs or spaces for indentation. This action applies to a selection, or the whole current file in the active editor.

To change indentation of a text fragment, do one of the following

- On the main menu, choose Edit | Indent Selection / Edit | Unindent Selection.
- Press `Tab` / `Shift+Tab`.

To fix indentation

Sometimes it is necessary to change indentation of a line at caret.

1. Place the caret at a line with wrong indentation.
2. Press `Ctrl+Alt+I`.

To toggle between tabs and spaces

- On the main menu, choose Edit | Convert Indents , and then choose To Spaces or To Tabs respectively.

Code Folding

On this page:

- Basics
- Code folding means
- Folding predefined and custom regions
- Commands of the Folding menu and associated shortcuts
- Specifying code folding preferences
- Viewing folded code fragments
  - Viewing the beginning of a folding region
- Using code folding comments
  - Supported folding comments
  - Surrounding a fragment with folding comments
  - Navigating to folding regions

Basics

You can collapse (fold) code fragments reducing them to a single visible line. In this way, you can hide the details that, at the moment, seem unimportant. If and when necessary, the folded code fragments can be expanded (unfolded).

Folded code fragments, normally, are shown as shaded ellipses (`...`).

Code folding means

You can collapse and expand code fragments by using:

- Code folding toggles (, or). These toggles are shown in the editor to the left of the corresponding folding regions. If a region is unfolded, indicates the beginning of the region while is located at its end. For folded regions, the toggle is shown as .
  If you hold the `Alt` key and click , or , the region is collapsed or expanded recursively, i.e. along with all its subordinate regions.
- Commands of the Folding menu and associated keyboard shortcuts. The Folding menu can be accessed from the main menu bar (Code | Folding), or as a context menu in the editor. The shortcuts are shown right in the menu. See Commands of the Folding menu and associated shortcuts .
- Folded fragments themselves: click `...` to expand the corresponding fragment. See also, Viewing folded code fragments .

Folding predefined and custom regions

You can fold and unfold:

- Code blocks, i.e. code fragments surrounded by a matched pair of curly braces `{}` .
  To collapse a code block, place the cursor within that block and then select Code | Folding | Fold Code Block or press `Ctrl+Shift+Period` .

  As a result, a custom folding region is formed, and the folding toggles for it appear. After that, the region can be collapsed and expanded as any other folding region.

  To remove a custom folding region, use the Fold Selection / Remove Region command for that region (`Ctrl+Period`).
- Predefined regions that correspond to such elements as import declarations, method bodies, anonymous and inner classes, documentation comments, etc. The predefined regions, roughly, correspond to the ones listed under Collapse by default on the Editor | General | Code Folding page in the Settings/Preferences dialog.
  For the predefined regions, the folding toggles are available right away, without the need to perform any additional actions.
- Any selected code fragment. A custom folding region for a selection is created and removed by means of the Fold Selection/ Remove Region command (`Ctrl+Period`).
- Regions surrounded by corresponding commented folding markers (e.g. `//<editor-fold desc="Description">...//</editor-fold>`). See Using code folding comments .

Note Code folding works for the keywords `if` / `while` / `else` / `for` / `try` / `except` / `finally` / `with` in case of at least two statements.

Commands of the Folding menu and associated shortcuts

The Folding menu can be accessed from the main menu bar (Code | Folding), or as a context menu in the editor.

Command	Shortcut	Description
Expand	`Ctrl+NumPad Plus`	Expand the current collapsed fragment
Collapse	`Ctrl+NumPad -`	Collapse the current folding region
Expand Recursively	`Ctrl+Alt+NumPad Plus`	Expand the current folded fragment and all the subordinate collapsed folding regions within that fragment
Collapse Recursively	`Ctrl+Alt+NumPad -`	Collapse the current folding region and all the subordinate folding regions within it
Expand All	`Ctrl+Shift+NumPad Plus`	Expand all collapsed fragments within the selection, or, if nothing is selected, expand all the collapsed fragments in the current file
Collapse All	`Ctrl+Shift+NumPad -`	Collapse all folding regions within the selection, or, if nothing is selected, collapse all the folding regions in the current file
Expand to level | 1, 2, 3, 4 or 5	`Ctrl+NumPad *, 1` `Ctrl+NumPad *, 2`	Expand the current fragment and all the nested fragments up to the specified level

	Ctrl+NumPad *, 3 Ctrl+NumPad *, 4 Ctrl+NumPad *, 5	
Expand all to level | 1, 2, 3, 4 or 5	Ctrl+Shift+NumPad *, 1 Ctrl+Shift+NumPad *, 2 Ctrl+Shift+NumPad *, 3 Ctrl+Shift+NumPad *, 4 Ctrl+Shift+NumPad *, 5	Expand all the collapsed fragments in the file up to the specified nesting level
Expand doc comments		Expand all documentation comments in the current file
Collapse doc comments		Collapse all documentation comments in the current file
Fold Selection / Remove region	Ctrl+Period	Collapse the selected fragment and create a custom folding region for it to make it "foldable" / Expand the current fragment and remove the corresponding custom folding region to make the fragment "unfoldable"
Fold Code Block	Ctrl+Shift+Period	Collapse the code fragment between the matched pair of curly braces `{}` and create a custom folding region for that fragment to make it "foldable"

Specifying code folding preferences

You can specify:

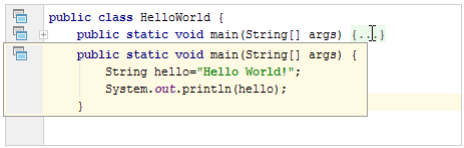
- Whether the code folding toggles should be shown.
- Which folding regions should be collapsed by default.

The corresponding settings are in the Settings dialog (Ctrl+Alt+S) on the Editor | General | Code Folding page.

For more information, see Code Folding page .

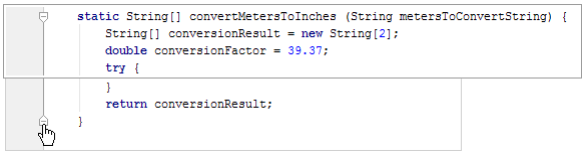
Viewing folded code fragments

To see the contents of a folded fragment, point to the ellipsis ... that indicates that fragment.



Viewing the beginning of a folding region

To see the beginning of a folding region - if it's not currently visible - point to the folding toggle at the end of that region.



Using code folding comments

- Supported folding comments
- Surrounding a fragment with folding comments
- Navigating to folding regions

Supported folding comments

You can create custom folding regions by surrounding code fragments with the commented lines. So doing, the comments can be either NetBeans style, or Visual Studio style.

NetBeans style

```
//<editor-fold desc="Description">
                    ...
                    //</editor-fold>
```

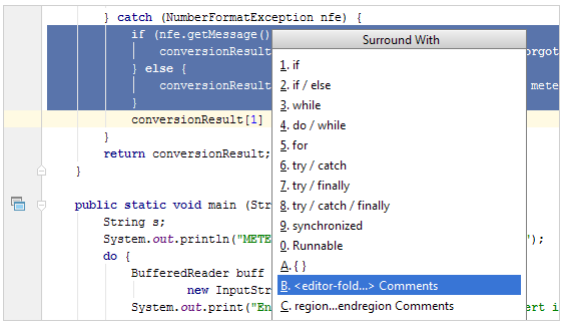
Visual Studio style

```
//region Description
                    ...
                    //endregion
```

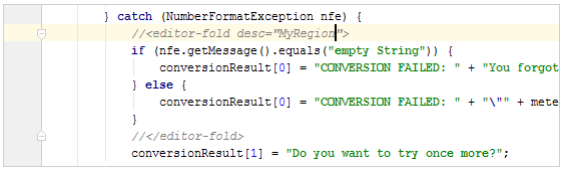
Once you have chosen the style for a file, don't use the other style in that file.

Surrounding a fragment with folding comments

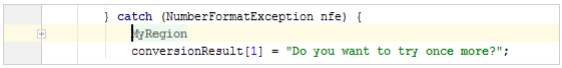
1. Select the code fragment of interest.
2. Select Code | Surround With or press `Ctrl+Alt+T`.
3. Select the folding comments to be used.



4. Specify the fragment description.



Now if you collapse the fragment, the description you have specified is shown in place of the code.



Navigating to folding regions

You can navigate to custom folding regions that were formed by surrounding code fragments with the corresponding commented folding markers:

1. Select Navigate | Custom Folding or press `Ctrl+Alt+Period`.
2. Select the target folding region. (The regions in the list are identified by their descriptions.)

Viewing Current Caret Location

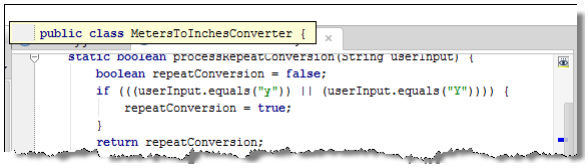
If in course of editing, searching, or navigating through a file, the cursor position runs out of the visible editor area, above the upper editor edge, you don't need to scroll through the file to obtain instant information about the current caret location.

To view the current caret position, do one of the following

– On the main menu, choose View | Context Info

– Press `Alt+Q`.

The pop-up frame appears on top of the editor, showing the name of the class or method where the caret currently resides:



Toggling Writable Status

If a file is read-only, it is marked with the closed lock icon in the Status bar , in its editor tab, or in the Project tool window. If a file is writable, then it is marked with the open lock icon in the Status bar. This section describes how to change read-only attribute of a file.

Tip If a read-only status is set by a version control system, it's suggested that you use IntelliJ IDEA version control integration features. For more information, see Version Control with IntelliJ IDEA.

To toggle read-only attribute of a file

1. Open file in the editor, or select it in the Project tool window .
2. Do one of the following:
   - On the main menu, choose File | Make File Read-Only, or File | Make File Writable.
   - Click the lock icon in the Status bar .

Toggling Case

On this page:

– Toggling between upper and lower cases
– Tips and tricks

Toggling between upper and lower cases

To toggle between upper case and lower case

1. Select text fragment, or just place the caret at the line you want to change case in.
2. On the main menu, choose Edit | Toggle Case, or press `Ctrl+Shift+U`.

Tips and tricks

Did you know that

– Applying Toggle Case to CamelCase name format converts the name format to lower case?
– The lowercase names are converted to the uppercase format?
– Applying Toggle Case to the uppercase name format converts the name format to lower case?

Highlighting Braces

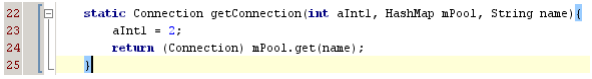
This editor feature significantly improves readability of the code, and simplifies search for unclosed blocks or tags.

To highlight block borders

- Place the caret immediately after the block closing brace/bracket or before block opening brace/bracket.

If the editor can find the block border, its braces, brackets or tags are highlighted with blue and a blue outline appears in the gutter area.

If the opening brace, bracket or tag is currently out of sight, you don't need to scroll to the beginning of the block. The editor shows a pop-up window on top that displays the beginning of the block.



If the editor cannot find the pair brace, bracket or tag, the unmatched one is highlighted with pink when the caret is placed next to it, and is underlined with a red curly line.

Spellchecking

On this page:

- Basics
- Checking the spelling of a word
- Configuring the dictionaries to use
- Configuring spellchecking options

Basics

IntelliJ IDEA helps you make sure that all your source code, including textual strings, comments, and literals, and commit messages, is spelt correctly. For this purpose, IntelliJ IDEA suggests a dedicated Typo inspection, which is supported by the corresponding bundled plugin and is enabled by default.

Correctness of spelling is checked against pre-defined dictionaries (as of now, `jetbrains.dic` and `english.dic`), and any number of user-defined custom dictionaries.

A user dictionary is a textual file with the `dic` extension, containing the words you want to be accepted by the Typo inspection as correct. The words in such dictionaries are delimited with the newline.

Besides that, you can define your own list of words that will be skipped by the inspection. You can add words to this list "on-the-fly", or intentionally while setting up your spellchecker options.

With the Typo inspection enabled, IntelliJ IDEA detects and highlights words not included in dictionaries and user's words list. It up to the user to provide correct spelling, accept word as is, or disable inspection.

If a word is accepted, it will be added to the user's words list, and skipped by the spellchecker in future. If inspection is disabled, all typos will be ignored.
In the textual strings and comments, spelling of a word at caret can be changed to a correct one. In the contexts that enable Rename refactoring, the inspection suggests to rename all occurrences of a symbol.

Checking the spelling of a word

1. Place the caret on a word highlighted by the Typo inspection.
2. Press `Alt+Enter` to show the available intention actions.
3. Choose one of the following actions:
   - Change to: select the desired spelling of a textual string or comment from the suggestion list.
   - Save to dictionary: add word to the user's list and skip it in future.

Configuring the dictionaries to use

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Spelling under Editor.
2. Define the set of dictionaries to be used for spellchecking.
   1. Switch to the Dictionaries tab.
   2. The Dictionaries area in the bottom of the page shows a list of the dictionaries that can be used in spellchecking. The list contains the dictionaries that come bundled with IntelliJ IDEA by default and user-defined dictionaries detected in the folders from the Custom Dictionaries Folder area above.
      - To have a default dictionary applied in the current project, select the check box next to it.
      - To exclude a default dictionary from spellchecking within the scope of the current project, clear the check box next to it.
   3. In the Custom Dictionaries Folder area, configure your custom dictionaries to use. This area displays a list of directories that contain user-defined dictionary files (text files with the `dic` extension, containing words separated with a newline).
      - To add a new folder to the list, click + and choose the required folder in the Select Path Dialog dialog that opens .
        The full path to the folder is added to the Custom Dictionaries Folder list, and all the `*.dic` files found in this folder are added to the Dictionaries list.
      - To remove a folder from the list, select it and click —.
3. Besides configuring a custom dictionary, you can create your own Word List with the words that you want to be skipped during spell checking without being included in a custom dictionary.
   1. Switch to the Accepted Words tab.
   2. Create a Word List:
      - Click the + icon to open the Add New Word dialog box and specify a new entry there. CamelCase or snake_case are not supported. If you try to add a word that is already included in one of the spelling dictionaries, IntelliJ IDEA displays an error message The word <just typed word> is already in the dictionary.
      - To remove an item from the list, select it and click —.

Configuring spellchecking options

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Inspections under Editor.
2. On the Inspections page, that opens showing a list of inspection types, expand the Spelling node and click Typo in the central pane.
3. In the right-hand pane, configure the Typo inspection:
   - In the Options area, define the type of contents to be inspected by selecting or clearing the Process Code, Process Literals, and Process Comments check boxes.
   - In the Severity area, choose the inspection severity level and the scope to apply this level in.

Scratches

On this page:

- Basics
- Creating scratch files
- Creating scratch buffers
- Observing the available scratches
- Closing scratches
- Deleting scratches
- Changing the language of a scratch
- Renaming, copying and moving scratches
- Important notes about scratches

Basics

IntelliJ IDEA provides a temporary editor. You can create a text or a piece of code for search or exploration purposes. IntelliJ IDEA suggests two types of temporary files:

Scratch files

The scratch files are fully functional, runnable, debuggable, etc. They require a language to be specified and have an extension. The scratch files are created via `Ctrl+Shift+Alt+Insert`.

Scratch buffers

The scratch buffers are only intended for pure editing, and as such they do not require specifying a language and do not have an extension. The scratch buffers belong to `.txt` type by default.

This action has no dedicated shortcut, but you can configure one as described in the section Configuring Keyboard Shortcuts .

Buffer files are reused after creating 5 files. So doing, after reuse, the content and language are reset.

Creating scratch files

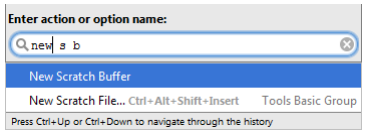
To create a scratch file

1. Do one of the following:
   - On the main menu, choose Tools | New Scratch File .
   - Press `Ctrl+Shift+Alt+Insert`
2. Select the language of the future scratch from the list that IntelliJ IDEA suggests. IntelliJ IDEA creates a temporary editor tab with the name `scratch.<extension>`. In the future, the default names will be `scratch_<number>.<extension>`.
3. Type the desired code.

Creating scratch buffers

To create a scratch buffer, follow these steps:

1. Press `Ctrl+Shift+A` or search everywhere .
2. Start typing the command name `New Scratch Buffer`:



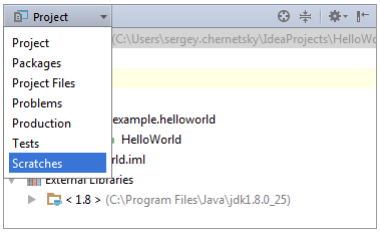
   IntelliJ IDEA creates a temporary editor tab with the name `buffer1`. In the future, the default names will be `buffer<number>`.
3. Type the desired code.

Note that though this action has no keyboard shortcut, you can still configure one as described in the section Configuring Keyboard Shortcuts . You can also switch from scratch files to scratch buffers by reassigning the shortcut to avoid garbage buildup.

Observing the available scratches

To observe the available scratch files and buffers, do one of the following:

- Choose Scratches view in the Project tool window :



- Press `Alt+F1` and choose Scratches (Navigating Between IDE Components).

Closing scratches

To close a scratch file or buffer, just click ✕ on the editor tab. Refer to the section Closing Files in the Editor for details.

Deleting scratches

To delete a scratch file or buffer, follow these steps:

1. Switch to the Scratches view of the Project tool window .
2. Under the `Scratches` pseudo-folder, right-click the scratch to be deleted, and choose Delete on the context menu.
3. Confirm deletion.

Changing the language of a scratch

If you want to change the scratch's language when a scratch is already created, you can do so with the aid of the editor's context menu. This is how it's done:

1. Switch to the Scratches view of the Project tool window , and open for editing the scratch file or buffer you want to change language for.
2. Right-click the editor background, and choose Change Language (<current language>) on the context menu.
3. Select the desired language.

   Note the following:

   - Four latest items appear on top of the list before a separator.
   - You can narrow down the list by typing the language name.
   - You can assign a shortcut to this action as described in the section Configuring keyboard shortcuts .
   - Change Language action keeps extension in sync, if it exists.

Renaming, copying and moving scratches

IntelliJ IDEA makes it possible to perform rename refactoring of the scratches. To rename a scratch, follow these steps:

1. In the Project tool window , switch to the Scratches view and select the scratch to be renamed.
2. Press `Shift+F6`.

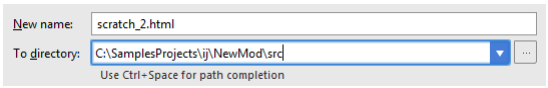
You can also perform renaming in the other ways:

- In NavBar : Jump to NavBar (`Alt+Home`) -> Rename (`Shift+F6`).
- In the Project tool window | Scratches view : Select In | Project | Scratches (`Alt+F1`) -> Rename (`Shift+F6`).
- Right from the editor: Refactor | Rename File.

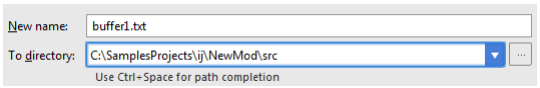
Copy and Move file actions are available the same way. Refer to the sections Copy and Move Refactorings for details.

Note that when copying a scratch, IntelliJ IDEA includes the respective extension that corresponds to the file type. This is how it's done:

1. In the Project tool window , switch to the Scratches view and select the scratch to be copied.
2. Press `F5`. IntelliJ IDEA shows the following dialog box:



This dialog box shows the scratch name with the corresponding extension. Note that when you copy a scratch buffer , the extension is `.txt` :



Important notes about scratches

Note the following:

- The scratch code in scripting languages is executable: you can run and debug it.
- Local history for scratches is supported.
- It is possible to perform clipboard operations with scratches.
- The scratches are stored, depending on your operating system,
  - Under IntelliJ IDEA home, in the directory `config/scratches` (on Windows/*NIX)
  - `~ Library->Preferences-><IntelliJ IDEA>XX->scratches` (on OS X)
- You can undo or redo changes in scratches.

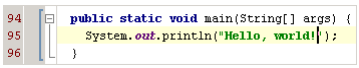
Adding, Deleting and Moving Code Elements

On this page:

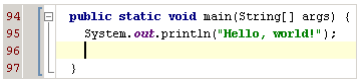
- Adding lines
- Duplicating lines
- Deleting lines
- Moving lines
- Moving statements
- Moving code element left or right

To add a line

- Press `Shift+Enter` to add a new line after the one where the caret is currently located and move the caret to the beginning of this new line.
  For instance, you have typed some text:



  Press `Shift+Enter` to start the next line immediately:



- To add a line before the current one, press `Ctrl+Alt+Enter`.

Tip Make sure that keyboard shortcuts are not in conflict. You can do that in the Keymap page of the Settings/Preferences dialog .

To duplicate a line or fragment

1. Place the caret at the line to be duplicated, or select the desired fragment of text.
2. Press `Ctrl+D`.

To remove a line

- Press `Ctrl+Y` to delete the line at caret.

To move a line

1. Place the caret at the line to be moved.
2. Do one of the following:
   - On the main menu, choose Code | Move Line Up or Code | Move Line Down.
   - Press `Shift+Alt+Up` or `Shift+Alt+Down`.

   IntelliJ IDEA moves the selected line one line up or down, performing the syntax check. For example:

```
public static void main(String[] args) {
    System.out.println("HW!!!");
}
```

   After moving line at caret:

```
System.out.println("HW!!!");
public static void main(String[] args) {
}
```

To move a statement up or down

1. Select a statement to be moved, or just place the caret at the first or last lines of a multi-line statement. Note that if moving a statement is not allowed in the current context, the commands will be disabled.
2. Do one of the following:
   - On the main menu, choose Code | Move Statement Up/Move Statement Down.
   - Press `Ctrl+Shift+Up` or `Ctrl+Shift+Down`.

   If you apply the same commands to a line at caret, or a to a selection, it will be moved one line up or down.

   IntelliJ IDEA moves the selected statement above the previous one, or directly underneath the next one, performing the syntax check. For example, place the caret at the method declaration:

```
public static void main(String[] args) {
    System.out.println("HW!!!");
}
public int foo(int a,int b){
    return a + b;
}
```

   After moving the statement:

```
public int foo(int a,int b){
    return a + b;
}
public static void main(String[] args) {
    System.out.println("HW!!!");
}
```

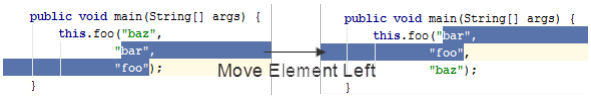
To move code element to the left or to the right

1. Place the caret at the desired code element, or select the elements to be moved.

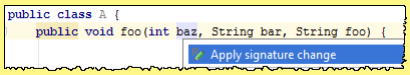
2. Do one of the following:
   - On the main menu, choose the commands Code | Move Element Left or Code | Move Element Right
   - Press `Ctrl+Shift+Alt+Left` or `Ctrl+Shift+Alt+Right`

Examples of code elements for which this functionality is currently implemented:

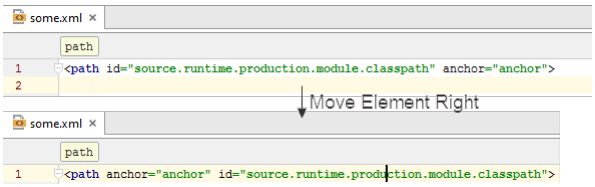
- Java: method invocation arguments, method declaration arguments, enum constants, array initializer expressions.



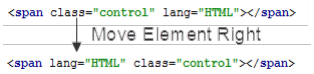
Tip You can propagate such a move in a method declaration to the method invocations - just immediately after move press `Alt+Enter`, and choose the action Apply signature change:



- XML: tag attributes.



- HTML: tag attributes.



Joining Lines and Literals

IntelliJ IDEA makes it possible to concatenate two unselected or several selected lines into one, removing the extra spaces, and providing the proper syntax. This operation smartly analyzes the lines being joined and treats them accordingly. For example, you can join lines of code, lines of comments, field declaration and initialization.

On this page:

– Joining lines

– Joining string literals

– Examples

Joining lines

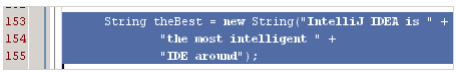
1. Place the caret on the line, to which the other lines should be added.

2. Sequentially press `Ctrl+Shift+J` keyboard shortcut, until all fragments are joined in a single line.

Tip You can select the lines and press `Ctrl+Shift+J` to obtain the same result.

Joining string literals

1. Select the lines with string literals that should be joined.



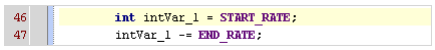
2. Press `Ctrl+Shift+J` keyboard shortcut. All redundant characters (spaces, quotes, and plus signs) are gone.

Examples

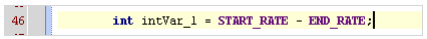
Joining a field or variable declaration and assignment:

Pressing `Ctrl+Shift+J` produces the following result, with the unwanted spaces and variable name in the second line removed:

Consider the following pair of statements:



Press `Ctrl+Shift+J` to join these lines into a correct single-line statement.

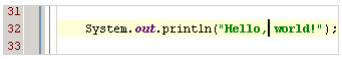


Splitting Lines With String Literals

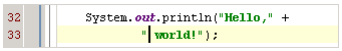
This feature is designed to correctly split string literals, providing the necessary quotation marks and plus signs.

To split a string literal into two parts:

1. Set the caret in the string literal to be split.



2. Press Enter.



Splitting String Literals on Newline Symbols

You can split a string literal on newline symbols (`\n`) by using the Break string on '\n' intention action . That is, a string literal in a code fragment like this:

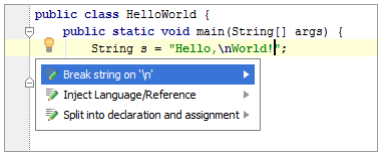
```
String s = "Hello,\nWorld!";
```

can be easily transformed into corresponding concatenation:

```
String s = "Hello,\n"
                    +
                    "World!";
```

To do that:

1. Place the cursor within the literal of interest.
2. Click the yellow light bulb icon or press `Alt+Enter` , and select Break string on '\n'.



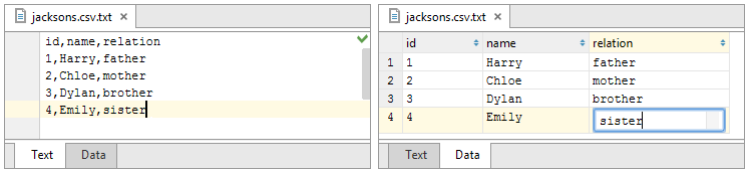
Editing CSV and Similar Files in Table Format

On this page:

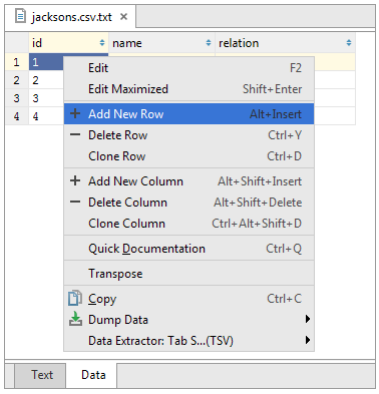
– Overview
– Prerequisites
– Opening the table editor
– Sorting data
– Hiding and showing columns
– Transposing the table
– Enabling coding assistance for a column
– Modifying cell contents
– Adding and deleting rows and columns
– Copying data to the clipboard or saving them in a file
– Changing data output format

Overview

For text files containing delimiter-separated values (e.g. CSV, TSV), IntelliJ IDEA provides an alternative, table editor.



Most of the functions in the table editor are accessed as context menu commands. Many of the commands have keyboard shortcuts.



Note that the context menus for the header row and the rest of the table are different.

Prerequisites

For the table editor and associated features to be available:

– You should be using the Ultimate Edition of IntelliJ IDEA. (The corresponding functionality is not available in the Community Edition.)
– The Database Tools and SQL plugin must be enabled. (This plugin is bundled with the IDE and enabled by default.)
– The file name extension must be associated with the text file type. See e.g. File Types .

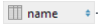
Opening the table editor

You can open the table editor for a whole file or for its fragment.

1. Open the file of interest in the editor.
2. If you want to open the table editor for a fragment, select that fragment.
3. Select Edit as Table from the context menu.
4. In the dialog that opens , specify conversion setting and click OK.

Sorting data

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: 

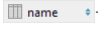
. The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance:

. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order:

.

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: 

.

Hiding and showing columns

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
   - Right-click any of the cells in the header row and select Column List.
   - Press `Ctrl+F12`.

   In the list that appears, the names of hidden columns are shown struck through.
2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

Transposing the table

The transposed table view is available. In this view, the rows and columns are interchanged.

To turn this view on or off, use the Transpose context menu command.

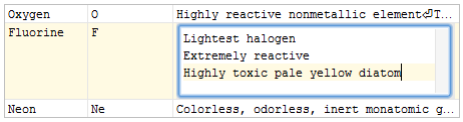
Enabling coding assistance for a column

You can assign a column one of the supported languages (e.g. SQL, HTML or XML): right-click the corresponding header cell, select Edit As and select the language. As a result, you get coding assistance for the selected language in all the cells of the corresponding column.

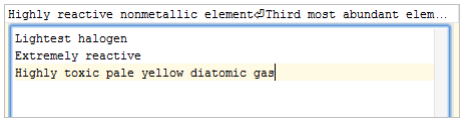
You can also assign a language to an individual cell .

Modifying cell contents

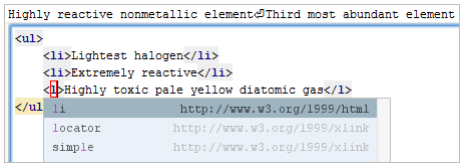
1. To start editing a value, do one of the following:
   - Double-click the corresponding table cell.
   - Right-click the cell and select Edit Value or Edit Maximized from the context menu.
   - Select the cell and press `F2` or `Shift+Enter`. In the latter case, the cell will be maximized.
   - Select the cell and start typing. Note that in this case the initial cell contents are deleted right away and is replaced with the typed value.
2. When in the editing mode, you can:
   - Modify the value right in the cell. To start a new line, use `Ctrl+Enter`. To save the changes, press `Enter`. To restore an initial value and quit the editing mode, press `Escape`.



   - Maximize the cell if you need more room for editing. To do that, press `Ctrl+Shift+Alt+M`, or right-click the cell and select Maximize.
     When working in a maximized cell, use `Enter` to start a new line and `Ctrl+Enter` to save the value. To restore an initial value and quit the editing mode, press `Escape`.



   - Insert the contents of a text file into the cell. To do that, right-click the cell and select Load File. Then select the necessary file in the dialog that opens.
   - Edit a value in the cell as a fragment in one of the supported languages (e.g. SQL, HTML or XML). To do that, right-click the cell, select Edit As and select the language. As a result, you get coding assistance for the language you have selected.



Adding and deleting rows and columns

Use the following context menu commands and shortcuts:

- Add New Row (`Alt+Insert`).
- Delete Row (`Ctrl+Y`). To delete more than one row at once, first, select the corresponding rows or cells in the corresponding rows.
- Clone Row (`Ctrl+D`). This command creates a copy of the current row.
- Add New Column (`Shift+Alt+Insert`).
- Delete Column (`Shift+Alt+Delete`). To delete more than one column at once, first, select the cells in the corresponding columns.
- Clone Column (`Ctrl+Shift+Alt+D`). This command creates a copy of the current column.

Copying data to the clipboard or saving them in a file

1. Use one of the following context menu commands:
   - Copy (`Ctrl+C`). This command copies the data for the selected cells to the clipboard.
     If the table is shown transposed (see Transposing the table), the range of cells is copied transposed (as shown) if the Allow Transposition option is on and non-transposed (as in the original table) otherwise.

     To turn the Allow Transposition option on or off, right-click the table, point to Data Extractor: <current_format> and click Allow Transposition.
   - Dump Data | To Clipboard. This command copies the data for the whole table to the clipboard.
   - Dump Data | To File. This command saves the data for the whole table in a file. Before actually saving the data, the dialog is shown which lets you select the output format and see how your data will look in a file.
2. If you are saving the data in a file, specify the file name and location.

See also, Changing data output format.

Changing data output format

When you copy table data to the clipboard (the Copy and Dump Data | To Clipboard commands), unless you change the format, the data are output as Tab-Separated Values (TSV).

To change the output format or to define a new format for the output:

1. Right-click the table and point to Data Extractor: <current_format>.
2. Do one of the following:
   - To select a different output format, click the name of the format.
   - To define a new delimiter-separated values format, or to view or modify the settings for an existing one, click Configure CSV Formats. As a result, the CSV Formats Dialog will open.

Using Drag-and-Drop in the Editor

IntelliJ IDEA allows copying and moving code fragments within the active editor tab, by means of drag-and-drop.

To move or copy code fragment

1. Make sure that using drag-and-drop is enabled in the Editor page of the IDE Settings.
2. Select the desired fragment in the editor.
3. Move or copy selection:
   - Move: Drag the selected fragment to the target location.
   - Copy: Keeping the `Ctrl` key pressed, drag selection to the target location.

Using Macros in the Editor

Macros provide a convenient way to automate repetitive procedures you do frequently while writing code. You can record, edit and playback macros, assign them a shortcut, and share them. Generally speaking, macros are designed for rather simple operations, and as such have the following limitations:

- Macros can be used for editor-related actions within a file.
- You cannot record such actions as button clicks, navigating to pop-up dialog boxes, and accessing tool windows or menus.

If a macro is intended for temporary use only, it is unnamed; permanent macros have unique names.

This section describes how to:

- Record Macros
- Bind Macros With the Keyboard Shortcuts
- Play Back Macros
- Edit Macros

To bind a macro with a keyboard shortcut

1. Open the Settings dialog and click Keymap .
2. Create a new keymap or select an editable keymap from the list of keymaps.
3. Expand the Macros node and select the macro for which a keyboard shortcut should be created.
4. Right-click on the macro and choose Add Keyboard Shortcut in the context menu.
5. In the Enter Keyboard Shortcut dialog, press the keys to be used as a shortcut. The keystrokes are immediately reflected in the First Stroke field. Optionally, select the Second stroke check box and specify the second stroke. As you press the keys, the Preview field displays the keystrokes you pressed, and the Conflicts field displays warnings, if the keystrokes are already in use.
6. Click OK using the mouse pointer to create a shortcut and bind it with the macro.

   Tip It is important that you use the mouse pointer, because any keystroke is interpreted as a shortcut.

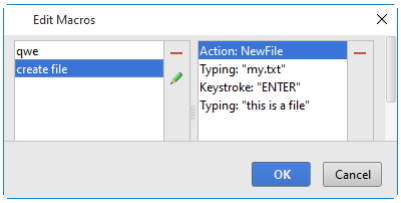
7. Click Apply to save the settings.

Editing Macros

After recording, you can remove or rename any or all of the macros from the list of available macros. The actions list of each macro is also editable - you can remove unnecessary actions.

To edit macros

1. On the main menu, choose Edit | Macros | Edit macros.
2. In left-hand pane of the Edit Macros Dialog , select the macro to be edited or deleted:



3. To delete a macro, click —. To change the macro name, click , and specify the new name in the Rename Macro dialog.
4. To change the list of actions for a macro, select an action in the action list, and click —.

Playing Back Macros

You can play back the recorded macros using the Edit | Macros menu commands, or custom shortcuts .

To play back a temporary macro

– On the main menu, choose Edit | Macros | Play Back Last Macro.

To play back a named macro

– On the main menu, choose Edit | Macros | <Macro name>.

To play back a macro with a keyboard shortcut

1. Select a keymap with the macro bindings on the Keymap settings page.
2. Press keyboard shortcut that corresponds to the desired macro.

To record a macro

1. On the main menu, choose Edit | Macros | Start Macro Recording. From that moment on, all your recordable actions are recorded.
2. When you are done with the procedure, choose Edit | Macros | Stop Macro Recording.
3. In the Enter Macro Name dialog, specify the name of the new macro, and click OK. If the macro is intended for temporary use only, you can leave the name blank.

Managing Editor Tabs

In this section:

- Managing Editor Tabs
  - Introduction
- Configuring Behavior of the Editor Tabs
- Navigating Between Editor Tabs
- Pinning and Unpinning Tabs
- Splitting and Unsplitting Editor Window
- Detaching Editor Tabs
- Editing Multiple Files Using Groups of Tabs
- Changing Placement of the Editor Tab Headers
- Sorting Editor Tabs

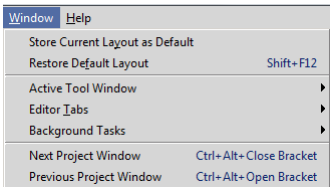
Introduction

Every time you open a file for editing, a dedicated tab is added to the editor window, next to the active editor tab.

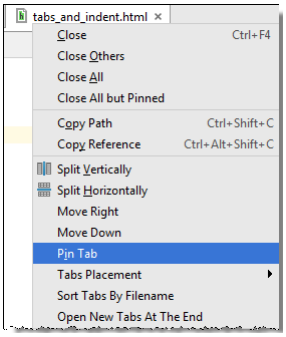
IntelliJ IDEA can limit the number of tabs opened in the editor simultaneously. When the number of tabs reaches its limit, the editor closes the tabs according to the tab closing policy, that is defined in the Editor Tabs settings page. By default, the tab limit is 10, but you can change it if necessary.

All commands, related to managing editor tabs, are available from:

- Window | Editor Tabs menu.



- Context menu of a tab.



Configuring Behavior of the Editor Tabs

On this page:

- Changing the number of editor tabs
- Disabling editor tabs
- Tips and tricks

Changing the number of editor tabs

To change the maximum allowed number of tabs

1. Open the Editor settings. To do that, click on the main toolbar, then click the Editor node in the Settings dialog box that opens.
2. In the Editor Tabs page of the Editor settings, type the desired maximum allowed number of the editor tabs to be opened at a time in the Tab limit field.

Disabling editor tabs

To disable editor tabs

- In the Editor Tabs page of the Editor settings, select None from the Tab placement drop-down list.

Tips and tricks

- If the tab limit equals to 1, the tabs will be disabled. If you want the editor to never close the tabs, type some unreachable number.
- With the disabled tabs, use the View | Recent files (`Ctrl+E`) command to quickly switch between files.

Navigating Between Editor Tabs

On this page:

- Navigating between editor tabs
- Navigating though the previously visited tabs
- Viewing all opened editor tabs and choosing the active editor

To navigate from the current tab to the next or previous tab

- Right-click the current editor tab and choose Select Next/Previous Tab on the context menu.
- Press `Alt+Right` or `Alt+Left`. So doing, the focus moves to the editor tab located next to the right or to the left from the active editor tab.
- Press `Ctrl+Tab` to use the Switcher .

This approach allows jumping from one tab to another as your editing session requires. While you move between the editor tabs, IntelliJ IDEA remembers the caret position within each opened file.

To go back and forth through the history of visited tabs

- On the main toolbar, click or .
- On the main menu, choose Navigate | Back / Forward.
- Press `Ctrl+Alt+Left` or `Ctrl+Alt+Right`.

This approach enables you to move back and forth through the history of your navigation, same way as it is done in a Web browser. As you move from file to file during your editing session, IntelliJ IDEA keeps track of the visited locations and enables you to go back, using the Navigate | Back / Forward commands.

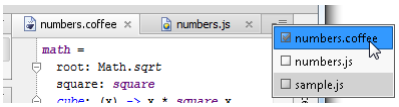
Note On an OS X computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.

To view all editor tabs and select the active editor

If all opened tabs are shown in a single row, some of the tabs may become invisible. IntelliJ IDEA helps display the list of the opened editor tabs that did not fit in the editor window and have become invisible whereupon you can choose the tab to activate.

1. Do one of the following:
   - On the main menu, choose Window | Editor Tabs | Show Hidden Tabs
   - Click

   The list of all the tabs that are opened but invisible appears. So doing the names of the tabs, which are currently visible, are displayed on the light background; the names of the tabs outside of the main window are shown on the darker background.
2. Click the name of the desired editor tab:



The selected editor tab becomes active and gets the focus.

The command Show All Tabs and the icon are only available when:

- The Show tabs in single row check box is selected in the Editor tabs page of the Editor settings.
- Some of the opened tabs are not visible because they do not fit in the editor window.

Pinning and Unpinning Tabs

On this page:

– Basics

– Pinning an editor tab

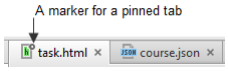
– Unpinning a tab

Basics

IntelliJ IDEA can limit the number of tabs opened in the editor simultaneously. When the number of tabs reaches its limit, the editor closes the tabs according to the tab closing policy, that is defined in the Editor Tabs settings page. By default, the tab limit is 10, but you can change it if necessary.

To prevent a tab from being closed automatically, you can pin this tab. Besides, when you close the editor tabs, you have an option to preserve pinned tabs opened and close only the unpinned tabs.

When a tab is pinned, there is a special marker on it.

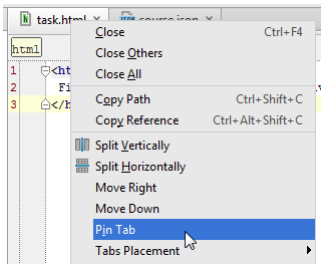


Tip The pinned status of a tab is helpful when you open a file for reference, rather than for editing.

Pinning an editor tab

1. Switch to the desired editor tab.

2. Right-click the editor tab, and choose Pin Tab on the context menu:



Unpinning a tab

1. Switch to the desired editor tab.

2. Right-click the editor tab, and choose Unpin Tab on the context menu.

Splitting and Unsplitting Editor Window

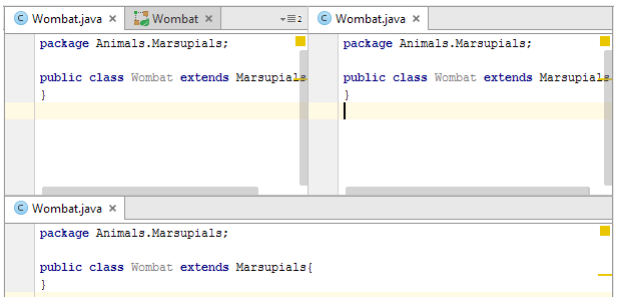
On this page:

- Basics
- Splitting editor tab
- Changing splitter orientation
- Removing a splitter

Basics

Splitting the editor window divides it into independent panes. You can split the editor window into as many panes as required, each one containing multiple tabs.

Each pane can be allocated vertically or horizontally. Thus, splitting helps create different editor layouts, organize tabs into groups, and edit multiple files simultaneously . For example, you can scroll through a part of a file, having at the same time the possibility to view the lines in its other part.



Splitting editor tab

To split an editor tab creating a file copy

1. Switch to the desired tab.
2. Right-click the tab header and choose Split Vertically or Split Horizontally on the context menu.

To split an editor tab without copying a file

1. Switch to the desired tab.
2. Right-click the tab header and choose Move Right or Move Down on the context menu.

Changing splitter orientation

To change splitter orientation

1. Switch to the desired tab.
2. Right-click the tab header and choose Change Splitter Orientation on the context menu.

Removing a splitter

To remove splitter

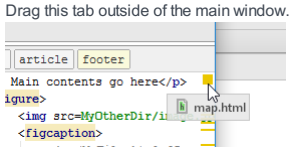
1. Switch to the desired tab.
2. Right-click the tab header and choose one of the following commands on the context menu:
   - To remove splitting in the active tab, choose Unsplit.
   - To remove splitting in all the open editor tabs, choose Unsplit All.

Detaching Editor Tabs

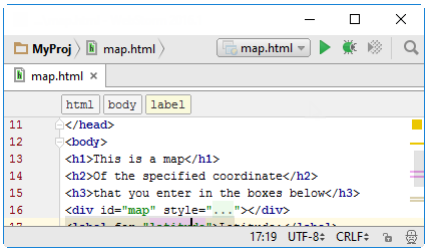
IntelliJ IDEA makes it possible to detach editor tabs, and move them to separate frames.

To detach an editor tab, do one of the following

– Drag this tab outside of the main window. A preview thumbnail appears:



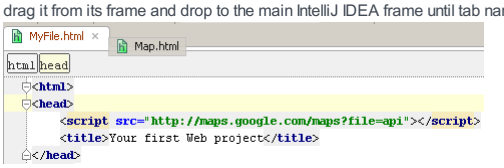
– Press Shift+F4.



The content of the editor tab opens in a separate frame.

To attach an editor tab

– drag it from its frame and drop to the main IntelliJ IDEA frame until tab name appears:



Tip The detached editor tabs can be split or unsplit.

– You can move editor tabs between split panes.

Editing Multiple Files Using Groups of Tabs

You can arrange tabs into groups to facilitate working with multiple files at a time. IntelliJ IDEA allows you to have an unlimited number of tab groups, thus enabling you to view several files or several places within the same file.

To create a new group of tabs

- Just split the desired editor tab.

To move a tab from one group to another

1. Switch to the desired tab.
2. Right-click the desired editor tab and choose Move Tab to Opposite Tab Group on the context menu.

Changing Placement of the Editor Tab Headers

By default, the tab headers appear on top of the editor, but you can change their location as required, and have the headers at the bottom, left, or right sides of the editor.

Note that the tab header placement is a global setting that applies to all projects.

To change location of the editor tab headers, do one of the following

- In the Editor Tabs settings page, select editor tab headers position from the Placement drop-down list.
- Right-click an editor tab, point to the menu item Tabs Placement, and then select the desired placement in the sub-menu.

Sorting Editor Tabs

On this page:

– Overview
– Enabling alphabetical sorting
– Sorting editor tabs alphabetically

Overview

The alphabetical sorting of the editor tabs is available regardless of the tabs position. However, alphabetical order for top and bottom placement becomes available, when the Show tabs in single row check command is selected on the Window | Editor Tabs | Tabs Placement menu.

Enabling alphabetical sorting

To enable alphabetical sorting:

1. On the main menu, choose Window menu.
2. Point to Editor Tabs | Tabs Placement.
3. Select the check command Show Tabs in Single Row.

Sorting editor tabs alphabetically

To sort editor tabs alphabetically:

1. Right-click an editor tab.
2. Select the check command Sort Tabs by Filename.
   If this check command is selected, the tabs headers are presented in alphabetical order. Otherwise, the editor tab headers appear in the opening order of the corresponding files.

Using TODO

In this section:

- Using TODO
  - Introduction
- Creating TODO Items
- Defining TODO Patterns and Filters
- Viewing TODO Items
- Tutorial: TODO Example
- Tutorial: Using Live Templates in TODO Comments

Introduction

Working on a large project, you often need to create the lists of tasks, and keep your team mates informed about the issues that require their attention. Such issues can include the questions that should be answered, certain changes that should be done later, areas of optimization and improvement etc.

IntelliJ IDEA suggests to use special TODO comments in the source code. Such comments can be used in all supported file types, and should match a certain TODO pattern. IntelliJ IDEA comes with one pre-defined pattern, but you can define as many TODO patterns as required. When a matching occurrence is encountered, it is interpreted as a TODO item. IntelliJ IDEA highlights such comments in accordance with the Colors and Fonts settings.

To create TODO items

1. Open the desired file in the editor and position the caret at the place where a TODO item should be created.
2. Create a comment . For example, you can use the `Ctrl+Slash` keyboard shortcut.
3. In the comment, type the string that matches one of your TODO patterns. By default, any string that starts with `TODO` (regardless of the case) is interpreted as a TODO item and is highlighted accordingly.
4. View the list of TODO items in the TODO tool window.

Defining TODO Patterns and Filters

On this page:

– Basics
– Defining TODO patterns
– Defining filters

Basics

TODO items in the source code are defined by a certain pattern.

Whenever a pattern is changed, or a new pattern is added, IntelliJ IDEA scans the whole project and rebuilds the index of TODO items. Results display in the TODO tool window , as described in the section Viewing TODO Items .

By default, IntelliJ IDEA provides two patterns:

– `\btodo\b.*`
– `\bfixme\b.*`

A generic pattern looks like `todo.*`

You might want to view the TODO comments of certain a type, and hide the others. For this purpose, IntelliJ IDEA suggests to use filters. This way you can show those items that match certain patterns only.

Defining TODO patterns

To define a TODO pattern, follow these general steps

1. Open the TODO page of the Settings dialog.
2. In the Patterns section, click the Add button + to create a new pattern, or the Edit button to update an existing one. The Add/Edit Pattern dialog opens.
3. In the Pattern field, enter the regular expression that describes the desired pattern.
4. In the Icon list, select the desired icon that will mark the matching TODO items in the TODO tool window .
5. Specify the color and font properties, which IntelliJ IDEA will use to highlight the matching comments in the source code.
6. Select the Case sensitive check box, if you want the pattern to be case-sensitive.

Defining filters

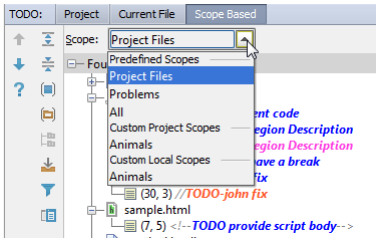
To define a filter that will be used to show specific types of TODO items, follow these general steps

1. Open the TODO page of the Settings dialog.
2. In the Filters section, click click the Add button + to create a new filter, or the Edit button to update an existing one.
3. In the Add/Edit Filter dialog, specify the filter name, and select the patterns to be included in the filter.

To view TODO items in project, follow these general steps

1. Open TODO tool window, as described in the procedure Showing a tool window .
   The tool window displays the encountered TODO items in several tabs:
   - All over the project (Project tab)
   - In the file currently active in the editor(Current File tab)
   - In one of the already defined scopes (Scope Based tab), which is quite useful for large projects.
   - In the current changelist, if version control support is enabled.

2. Click the desired tab (view), and explore the list of encountered TODO items. For example, with the Scope Based view selected, one has to choose scope from the drop-down list.



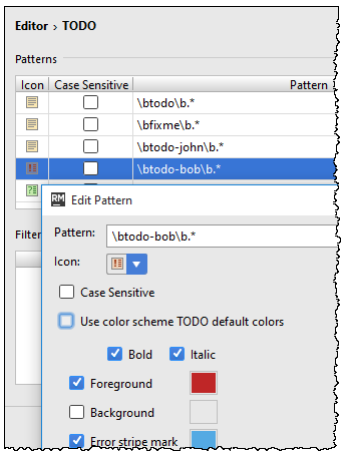
3. Narrow down the list of search results by choosing scope, and applying filters.

Tutorial: TODO Example

Consider the following example: creating and viewing TODO items for each of the team members.

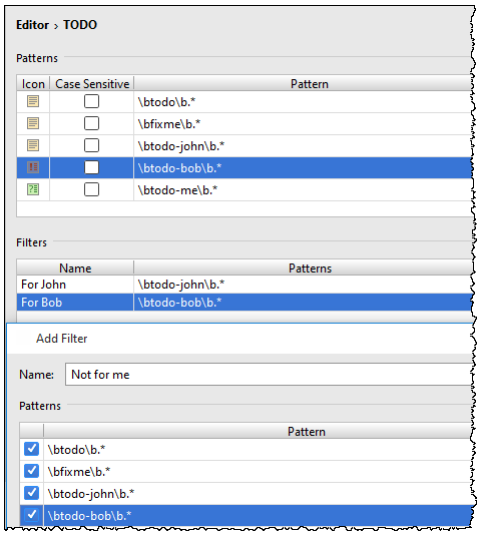
Creating Patterns for TODO Items

On the TODO page of the Settings dialog, click + in the Patterns section, and create additional TODO patterns , for example, `todo-John` , `todo-Bob` and `todo-me` , with new icons, and custom color schemes:



Creating Filters

Next, create several filters, which you will use to show the TODO items, say, for each of the developers, and not for your good self. For this purpose, in the Filters section, click +, and specify the filter names, for example, `For John` , `For Bob` , and `not for me` . Associate these filters with the patterns:



Creating TODO Items in Source Code

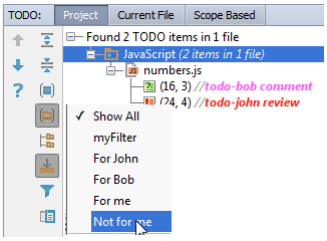
Now, in the source code, create TODO items: in the line of code, where you want to add a note, press `Ctrl+Slash` , or `Ctrl+Shift+Slash` , and type `TODO` that matches one of the patterns, followed by some meaningful description:

```
    //TODO-bob region Description
    math = {
        root: Math.sqrt,
        square: square,
        cube: function(x) {
            return x * square(x);
        }
    };
    race = function() {
        var runners, winner;
//      todo-me have a break

        //endregion
//TODO-john fix
        winner = arguments[0], runners = 2
```

Viewing TODO Items

Having produced a number of TODO items across the whole project, review them in the TODO tool window: click the TODO button on the tool window bar to show the tool window. By default, all the encountered TODO items are displayed.

Let's now show the TODO items for Bob and John, and hide the other items: click the filter icon on the toolbar of the TODO tool window, and select Not for me in the menu:


TODO:
Project
Current File
Scope Based
Found 2 TODO items in 1 file
JavaScript (2 items in 1 file)
numbers.js
(16, 3) //todo-bob comment
(74, 4) //todo-john review
Show All
myFilter
For John
For Bob
For me
Not for me

Tutorial: Using Live Templates in TODO Comments

On this page:

– Overview
– Creating TODO pattern and filter
– Creating live template and variables
– Using the REVIEW items

Overview

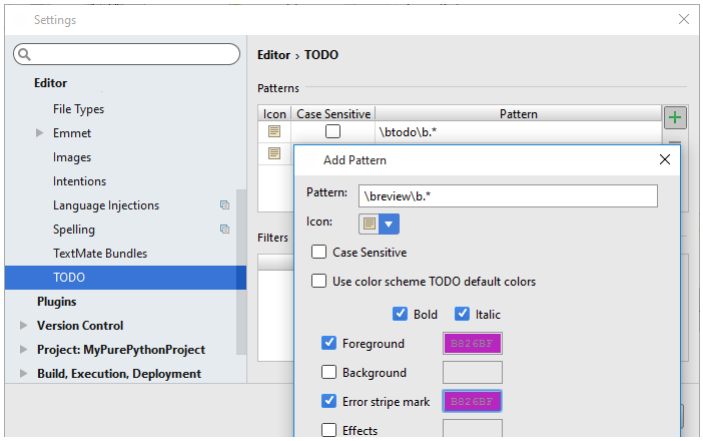
Let's explore an advanced IntelliJ IDEA's facility to create a live template for the TODO items' text. Why do we need it at all? For example, you want your team mates to create unified TODO items, with the user name automatically filled in, followed by some arbitrary text.

This is how it's done.

Creating TODO pattern and filter

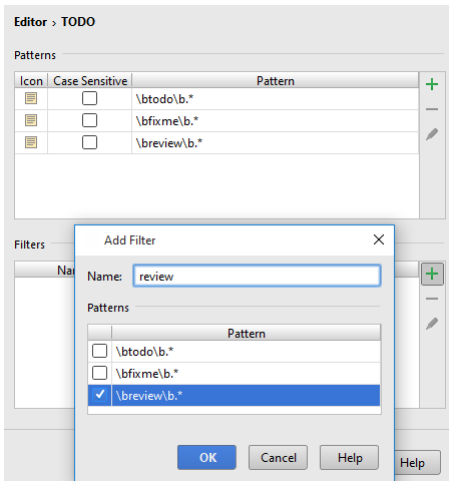
Open the Settings/Preferences dialog, and under the Editor section, click TODO .

Create pattern `review` . To do that, click + in the Patterns section:



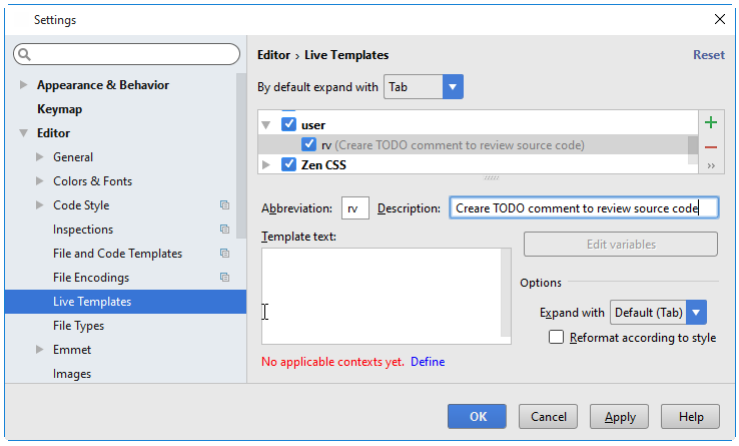
Define color in the Color Picker - in this case, it's pink.

Next, let's create a filter . To do that, click + in the Filters section, and define the filter:



Creating live template and variables

Next, back in the Settings/Preferences dialog, under the Editor section, click Live Templates .

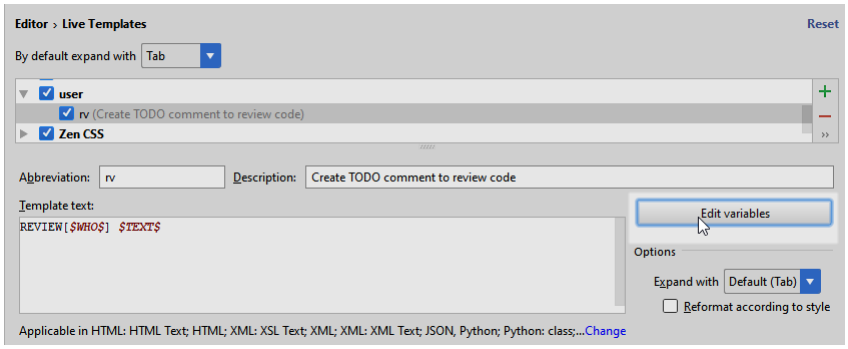


Note that the new template is added to the automatically created group `user`.

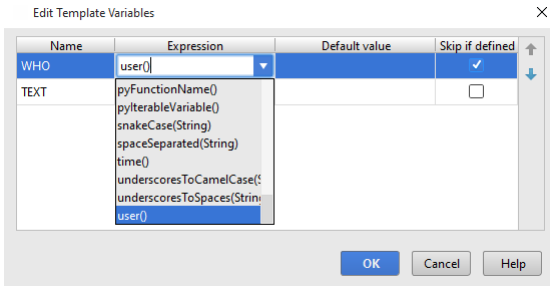
Next, pay attention to the red note at the bottom. It says that the new template lacks context, where it should apply. So let's click the link Define, and allow all possible contexts. And finally, let's define the body of the template itself: in the area Template text, type the following:

```
REVIEW[$WHO$] $TEXT$
```

We have two undefined variables here: `$WHO$` and `$TEXT$`. The variable `$TEXT$` will be used just as an input field, while the variable `$WHO$` should be filled in automatically. To define this variable, click the button Edit variables:



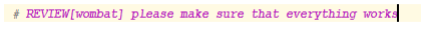
Next, in the Edit Template Variables dialog box, select an expression for the variable `$WHO$`:



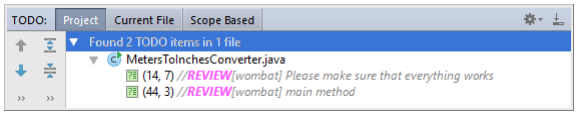
Using the REVIEW items

Now let's make sure it works. Back in the editor, create a line comment (`Ctrl+Slash`), type `rv`, and press `TAB`:

Note that a right-gutter stripe next to the TODO comment is also added to the editor. As you see, the live template `rv` has automatically populated the user name, leaving us with the task of just entering some meaningful comment:



Now, when you opt to show REVIEW comments only, use the filter. To do that, click and select the filter review to show those TODO comments only, that have the keyword REVIEW.



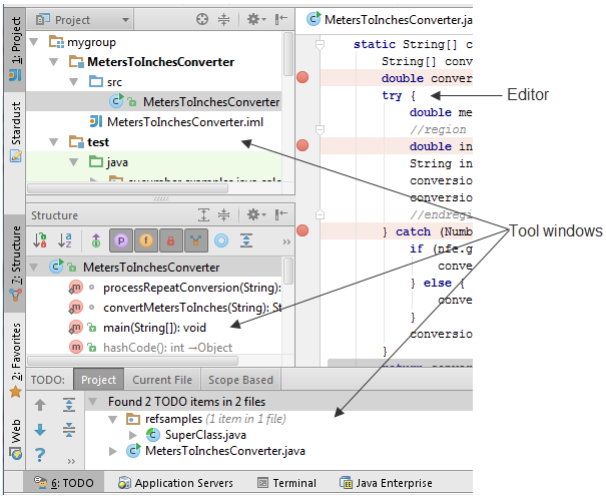
Tool Windows

In this section:

- Tool Windows
  - Overview
  - Tool window quick access
  - Tool window bars and buttons
  - General tool window layout
  - Accessing tool window menus
- Manipulating the Tool Windows
- Viewing Modes
- Speed Search in the Tool Windows
- Specifying the Appearance Settings for Tool Windows
- Managing Your Project Favorites

Overview

Attached to the bottom and sides of the workspace are IntelliJ IDEA tool windows. These secondary windows let you look at your project from different perspectives and provide access to typical development tasks. These include project management, source code search and navigation, running and debugging, integration with version control systems, and many other specific tasks.

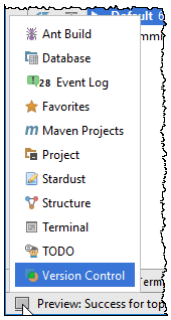


Certain tool windows are available always, that is, in any project irrespective of the project nature, contents, and configuration. Other tool windows are available only if the corresponding plugins and/or facets are enabled. There are also tool windows that require certain specific actions to be made to become available. (For example, to make the Find tool window available you have to search for something first.)

Tool window quick access

In the lower left corner of the workspace, there is a button which initially looks like this .

If you point to this button, a menu opens that provides quick access to tool windows.

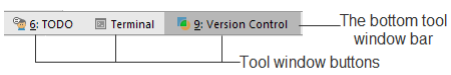


Option names in this menu correspond to the names of the tool windows. When you select an option, the corresponding tool window opens and becomes active.

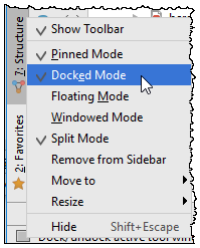
If you click the button, tool window bars and buttons are shown. At the same time the button appearance changes to . If you click the button again, the tool window bars and buttons are again hidden.

Tool window bars and buttons

When visible, the tool window button bars (or just tool window bars) are around the tool windows (or the editor area if the tool windows are hidden). These bars contain the buttons for showing or hiding the tool windows (tool window buttons).



The tool window buttons also provide access to tool window context menus which are shown when the buttons are right-clicked.



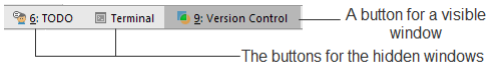
The context menus let you control the tool window viewing modes and other aspects of the tool window appearance.

Initially, there are three button bars, two at the sides of the main window and one at the bottom. You can show or hide all the button bars at once by clicking in the bottom-left corner of the workspace.

Each tool window button has the name of the corresponding tool window on it. On certain buttons, the window name may be preceded by a number, for example, 1: Project. This means that the keyboard shortcut `Alt+<number>` is available for showing or hiding the window. You can, for example, show or hide the Project tool window by pressing `Alt+1` .

You can turn showing the window access numbers on the buttons on and off in the Appearance settings .

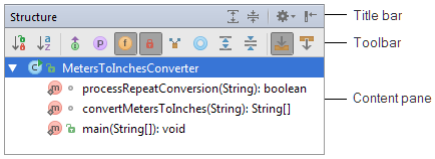
The buttons for visible tool windows and for hidden ones are shown differently.



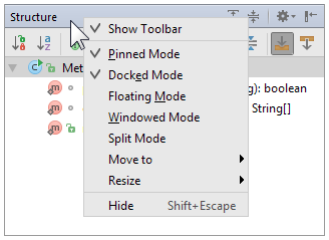
You can rearrange the tool windows by dragging-and-dropping the tool window buttons onto a different tool window bar (or to a different corner of the same bar). As a result, the tool window becomes attached to the bar you've moved the window button to.

General tool window layout

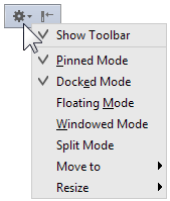
Generally, all the tool windows are organized in a similar way.



At the top of the window is a title bar. When you right-click the title bar, a menu for managing the window appearance and contents is shown.



The title bar contains two buttons in its right-hand part. The first of these buttons ( ) opens a menu for managing the tool window viewing modes . Note that the menu options are a subset of the title bar context menu.



The second of the buttons ( ) is for hiding the tool window. When used in combination with the `Alt` key, clicking this button hides all the windows attached to the same tool window bar.

Underneath the title bar are the toolbar and content pane. Depending on the window, the toolbar may be above or to the left of the content pane.

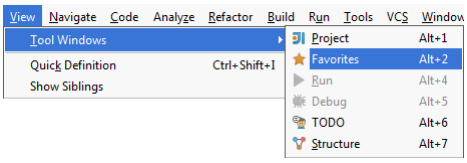
The toolbar buttons, generally, are window-specific. However, the windows with similar purposes may contain similar controls on their toolbars.

In most cases, a function associated with a toolbar button may also be accessed from the main or context menu, or may have a keyboard equivalent.

The content panes may be plain or contain two or more "layers" (views) represented, for example, by tabs. There are also tool windows with the content pane part shown on a separate tab in the editor area.

Accessing tool window menus

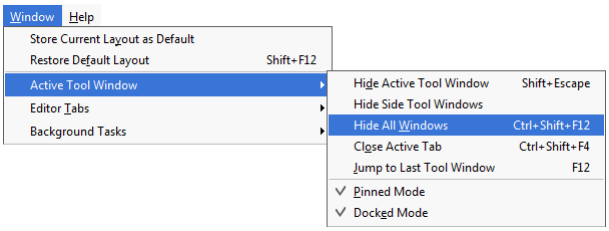
– Use the View | Tool Windows menu to show or hide the tool windows.



Warning! Note that some of the tool windows (for example, Find or Hierarchy tool window) appear in the View menu ONLY when an action has been performed for the first time. So, unless you do something, say, find or build a hierarchy, these tool windows are not visible in the View menu.

Same happens with the Messages and Problems tool windows. For example, the Messages tool window shows warnings from the compiler or the status after finishing a task, etc. Similarly, the Problems tool window appears only if auto make option is enabled and your code contains a compilation error.

– Use the Window | Active Tool Window menu to perform operations related to an active tool window. These include hiding the active and other windows, changing the viewing modes for the active tool window, and more.



Manipulating the Tool Windows

You can manipulate the tool windows in various ways to adjust IntelliJ IDEA workspace to your needs.

- Showing a tool window
- Hiding an individual tool window
- Hiding all tool windows attached to the same tool window bar
- Hiding all tool windows
- Switching to the last active tool window
- Hiding or showing the tool window bars
- Hiding tool window buttons
- Attaching a tool window to a different tool window bar
- Resizing a tool window
- Increasing the number of tool windows shown at a time
- Saving and restoring the arrangement of the tool windows

Showing a tool window

Do one of the following:

- In the lower left corner of the workspace, point to and select the tool window in the menu that is shown.
- Click the corresponding tool window button on the tool window bar .
- Choose View | Tool Windows | <tool window> in the main menu.
- If the tool window has an associated quick access number, press `Alt+<number>` (for example, `Alt+1` for the Project tool window).

Hiding an individual tool window

Do one of the following:

- Click the corresponding tool window button on the tool window bar .
- Click on the tool window title bar.
- Right-click the corresponding tool window button and select Hide from the context menu.
- Right-click the tool window title bar and select Hide from the context menu.
- Choose View | Tool Windows | <tool window> in the main menu.
- If the tool window has an associated quick access number, press `Alt+<number>` (for example, `Alt+1` for the Project tool window).
- If the tool window you are going to hide is currently active, press `Shift+Escape`.

Hiding all tool windows attached to the same tool window bar

Do one of the following:

- Press and hold the `Alt` key, and click on the title bar of any of the tool windows attached to the corresponding tool window bar.
- Choose Window | Active Tool Window | Hide Side Tool Windows in the main menu. This command hides all the tool windows attached to same tool window bar as the active tool window or the last of the active tool windows.

Hiding all tool windows

Do one of the following:

- Choose Window | Active Tool Window | Hide All Windows in the main menu.
- Press `Ctrl+Shift+F12`.

Switching to the last active tool window

Do one of the following:

- Choose Window | Active Tool Window | Jump to Last Tool Window in the main menu.
- Press `F12`.

If all the tool windows are currently hidden, the last active tool window will be shown and made active.

Hiding or showing the tool window bars

You can hide all the tool window bars if you need more space in the IntelliJ IDEA window:

- In the lower left corner of the workspace, click .

If the tool window bars are hidden, you can bring them back onto the screen either permanently or for a short period of time:

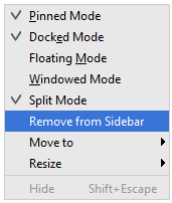
- To restore the tool window bars, click in the lower left corner of the workspace.
- To see the tool window bars for a short period of time, double-press and hold the `Alt` key. The tool window bars appear on the screen making the tool window buttons accessible. The tool window bars are hidden again when you release the key.

Hiding tool window buttons

IntelliJ IDEA makes it possible to hide individual tool window buttons, without actually uninstalling the corresponding plugins.

To remove a tool window button from view:

1. Make sure that the tool window bars are visible .
2. Right-click the tool window button you want to hide.
3. Choose Remove from Sidebar



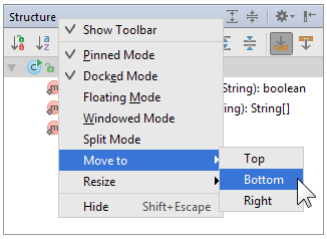
To restore the hidden tool window button, choose View | Tool Windows on the main menu, and then click the window with the hidden toolbar button.

Attaching a tool window to a different tool window bar

Do one of the following:

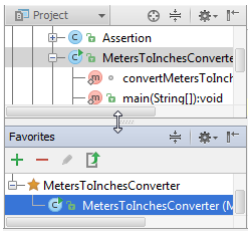
- Drag the corresponding tool window button onto the desired tool window bar (top, left, bottom or right).

- Right-click the corresponding tool window button or the tool window title bar to open the context menu. Choose Move to and then select the destination tool window bar (Top, Left, Bottom or Right).



Resizing a tool window

- Hover the mouse pointer over the tool window border. When the pointer becomes a double-headed arrow, drag the border in the required direction.



- You can also resize a tool window by moving its border to left or right, or up or down in steps. The following alternatives are available for doing that:
  - Right-click the corresponding tool window button or title bar and select Resize. Then select one of the available Stretch to options.
  - Make the tool window of interest active and do one of the following:
    - Choose Window | Active Tool Window | Resize, and then select the necessary Stretch to option.
    - Press `Ctrl+Shift` in combination with the corresponding arrow key.

Increasing the number of tool windows shown at a time

To increase the number of tool windows to be shown at a time, you should appropriately set the viewing modes for different tool windows. Consider the following:

- Generally, for a tool window to be visible always (i.e. even when inactive), the tool window should be pinned .
- There are no limiting factors for the number of visible floating windows and ones in the windowed mode. Note that the windowed mode is not available if you are using OS X.
- To be able to see two windows docked to the same tool window bar at a time (rather than one), one of the windows should have the split mode off and the other one on.
- Initially, three (out of four) tool window bars are used. You can "activate" the forth tool window bar (the top one) by moving certain tool windows to it.

Saving and restoring the arrangement of the tool windows

You can save the way the tool window are currently arranged by choosing Window | Store Current Layout as Default in the main menu.

At a later time, you can return to the saved workspace layout by choosing Window | Restore Default Layout (`Shift+F12`).

Viewing Modes

IntelliJ IDEA provides various viewing modes that let you control the way the tool windows are shown and behave. These modes help you keep a proper balance between quick, easy access to tool windows and maximum screen space for editing your code.

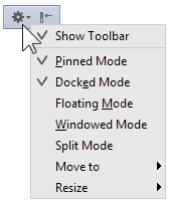
The viewing modes are set separately for each of the tool windows.

- Ways to control the viewing modes
- Fixed / floating / windowed mode
- Docked / undocked mode
- Pinned / unpinned mode
- Split mode
- Group Tabs option
- Wide screen support

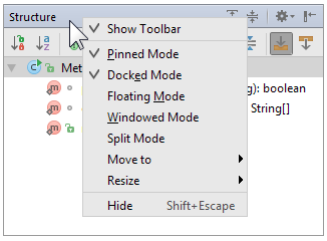
Ways to control the viewing modes

The viewing modes are set by turning the corresponding viewing options on or off. To access these options, you can use:

- on the title bar of a tool window.



- Context menus. The context menus are accessed by right-clicking the tool window buttons or the tool window title bars.

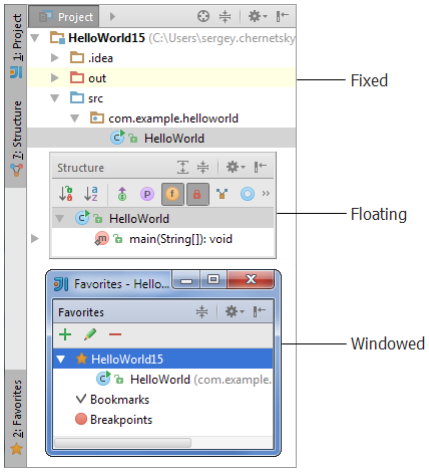


- For an active tool window: the Window | Active Tool Window menu.

Fixed / floating / windowed mode

A tool window may be fixed. In that case, it stays within the main window.

Alternatively, a tool window may be in the floating or in the windowed mode. Note that the windowed mode is not available if you are using OS X.



When in the fixed mode, one side of the tool window is attached to one of the tool window bars. The behavior of the opposite side depends on whether the window is docked or undocked .

Initially, all the tool windows are in the fixed mode (i.e. the floating and windowed modes are off).

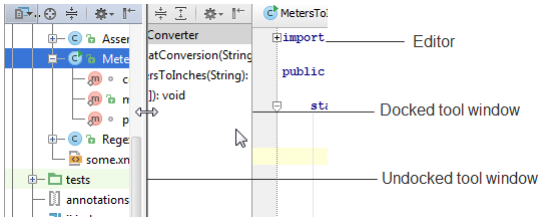
In the floating and in the windowed modes, a tool window may be moved around the screen to any place.

To switch to the floating or the windowed mode, turn on the Floating Mode or the Windowed Mode option. To bring a tool window back to the fixed mode, turn the Floating Mode and the Windowed Mode options off. See Ways to control the viewing modes.

Note that for the tool windows in the windowed mode, the Window menu command Hide Active Tool Window is disabled.

Docked / undocked mode

A tool window in the fixed mode may be docked or undocked.



In the docked mode, all the sides of a tool window are attached to surrounding elements (the editor, other tool windows, etc.) Thus, the tool window and the adjacent elements share the space available in the main window.

When a docked tool window becomes inactive, it stays visible or is hidden depending on whether the window is pinned or unpinned .

Initially, all the tool windows are in the docked mode (i.e. the docked mode is on).

When undocked, all the sides of a tool window (except the one at the tool window bar) are detached from surrounding elements. The window moves to the "upper layer" covering the elements it used to share the space with. In one of the directions (along the tool window bar), the window stretches and takes all the available space. In the other direction, one of the window borders becomes loose and can be moved without affecting the sizes of other, underlying elements.

When an undocked tool window becomes inactive, it is automatically hidden.

To switch between the docked and undocked mode, turn the Docked Mode option on or off. See Ways to control the viewing modes.

Pinned / unpinned mode

Pinned tool windows, generally, stay visible when becoming inactive. Unpinned tool windows in such cases are automatically hidden.

Initially, all the tool windows are pinned (i.e. the pinned mode is on).

There may be slight differences in behavior depending on the other viewing modes:

- Undocked tool windows are always hidden when inactive. (In the undocked mode, the tool windows are effectively unpinned.)
- Floating pinned tool windows, when inactive, may become semi-transparent.

To switch between the pinned and unpinned mode, turn the Pinned Mode option on or off. See Ways to control the viewing modes.

Split mode

This mode has to do with how many windows docked to the same tool window bar may be shown at a time (one or two).

Generally, the space along a tool window bar is shared between two groups of docked tool windows.

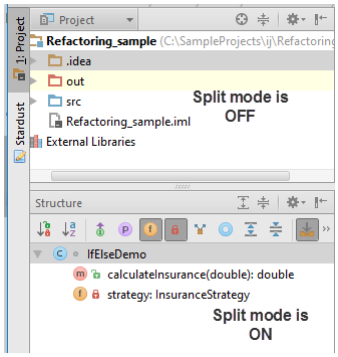
In one of the groups are the tool windows for which the split mode is off; in the other group are the ones with this mode on.

At each moment of time, only one window from each of the groups may be visible.

Thus, if all the tool windows docked to a tool window bar have the split mode off, only one tool window may be shown at a time. In this case, the tool window which is visible takes all the space available along the tool window bar. So when you make a certain window visible, the previous visible window is automatically hidden.

The same behavior is observed if the split mode is on for all the windows docked to the same tool window bar.

To be able to see two windows simultaneously, the corresponding windows should belong to different groups, that is, one of the windows should have the split mode off and the other one on.

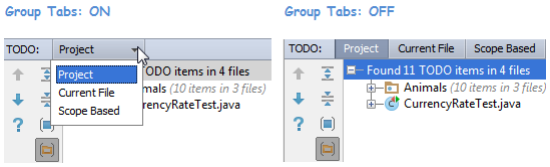


The tool window buttons for the tool windows with different settings of the split mode are grouped and shown at different corners of the corresponding tool window bar. For vertical window bars, the windows with the split mode off have the buttons at the top corner; for the horizontal bars, the buttons for such windows are at the left corner.

To turn the split mode on or off see Ways to control the viewing modes.

Group Tabs option

If more than one view is available in a tool window, the corresponding views may be shown on separate tabs if the Group Tabs option is off. If this option is on, the views are selected from a list.



Wide screen support

IntelliJ IDEA makes it possible for the tool windows to use the full width and height of the screen. In the Settings dialog, expand the node Appearance and Behaviour, and in the Appearance page, use the check boxes Wide screen tool window layout and Side by side layout on the left/right to optimize placement of the tool windows.

Note also that you can turn side-by-side layout on or off by `Ctrl+MouseClick` on splitter between the tool windows.

Refer to Appearance for details.

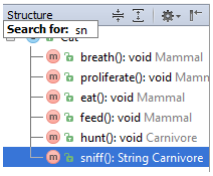
Speed Search in the Tool Windows

Speed search in the tool windows helps you find and navigate to a file or folder in the Project tool window, a member in the Structure tool window, a changelist in the Version Control tool window, an item in the TODO list, and more.

Note that speed search is performed only on epanded nodes, if a node is folded the matching items under it are not detected.

To search through a tool window

1. Select the desired tool window.
2. Start typing the item name (for instance, file, class, field, etc.). As you type, the Search for field appears over the tool window toolbar showing the entered characters, and the element selection moves to the first item that matches the specified string. The matching part of the string is highlighted.



3. If several neighboring items match the pattern, use the Up and Down keys on the keyboard to navigate among them.
4. Press `Enter` when ready. As a result, the matching item is selected in the tool window. Pressing `Escape` hides the Search for field.

Specifying the Appearance Settings for Tool Windows

You can change certain tool window appearance properties by specifying the corresponding Appearance settings.

To change the appearance properties for tool windows

1. In the Settings dialog, expand the Appearance&Behaviour node, and click Appearance .
2. If necessary, change the settings related to tool window appearance. These are mainly in the Transparency and the Window Options sections. For more information, see descriptions of the pages under Appearance and Behavior .

Managing Your Project Favorites

On this page:

- Basics
- Using the Project tool window to add items to favorites
- Using the editor to add items to favorites
- Creating a new favorites list
- Renaming a favorites list
- Moving an item to a different list
- Removing items from favorites

Basics

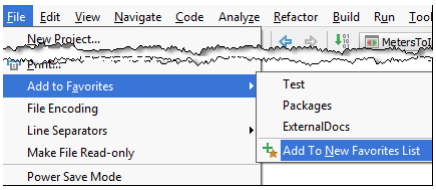
You can arrange the most frequently used project items (files, folders, packages, instance and class members, etc.), bookmarks , and breakpoints in the lists of favorite items (favorites). In IntelliJ IDEA, there is a dedicated tool window for managing your favorites (the Favorites tool window).

Initially, there is one (empty) favorites list which has the same name as the project.

You can create more favorites lists and manage their contents as necessary.

Using the Project tool window to add items to favorites

1. In the Project , select the item or items you want to add to favorites.
2. Do one of the following:
   - On the main menu, point to File | Add To Favorites.
   - On the context menu of the selection, point to Add To Favorites
3. To add the selected item or items to an existing favorites list, select the name of the list.
   To create a new favorites list and add the selected item or items to it, select Add To New Favorites List. In the Add New Favorites List dialog, specify the name of the new list and click OK.



Tip Add items to favorites using drag-and-drop: drag the item of interest from the Project tool window, or an external file from Explorer or Finder, and drop it onto the desired favorites list in the Favorites tool window.

Using the editor to add items to favorites

In the editor, you can add to favorites one file, or all the currently opened files.

1. Right-click the editor tab of interest and select one of the following options:
   - If you want to add the current file to favorites, select Add To Favorites. (For the current file, File | Add To Favorites is also available as an alternative.)
   - If you want to add all the files open in the editor to favorites, select Add All To Favorites.
2. To add the item or items to an existing favorites list, select the name of the list.
   To create a new favorites list and add the item or items to it, select Add To New Favorites List. In the Add New Favorites List dialog, specify the name of the new list and click OK.

Creating a new favorites list

You have an option of creating a new favorites list when adding items to favorites or when moving a favorites list item to a different list.

You can also create a new (empty) favorites list just on its own, as a separate task:

1. If the Favorites tool window is not currently shown, open it .
2. Do one of the following:
   - Click + on the toolbar.
   - Press `Alt+Insert`.
3. In the Add New Favorites List dialog, specify the name of the new list and click OK.

Renaming a favorites list

1. If the Favorites tool window is not currently shown, open it .
2. Do one of the following:
   - Right-click the list whose name you want to change and select Rename Favorites List.
   - In the toolbar of the Favorites tool window, click .
3. In the New Name for Favorites List dialog box, change the name of the list as required, and click OK.

Moving an item to a different list

1. If the Favorites tool window is not currently shown, open it .
2. Right-click the list item that you are going to move and select Send To Favorites.
3. To move the selected item to an existing favorites list, select the name of the destination list.
   To create a new favorites list and move the item there, select Send To New Favorites List. In the Add New Favorites List dialog, specify the name of the new list and click OK.

Tip To move an item from one favorites list to another, use drag-and-drop.

Removing items from favorites

To remove items from favorites, you can delete the corresponding favorites list items and/or the whole favorites lists.

1. If the Favorites tool window is not currently shown, open it .
2. Select the item or items that you want to remove from favorites. Note that you can select separate list items and the whole lists at the same time.
3. Do one of the following:
   - Click — on the toolbar.
   - Select Remove From Favorites in the context menu. (If a single favorites list is currently selected, note that there are also the following options: Delete Favorites List <list_name> and Delete All Favorites Lists Except <list_name>.)
   - Press `Delete`.

Creating and Managing Projects

In this section:

- Creating Projects
- Opening, Reopening and Closing Projects
- Opening Multiple Projects
- Working with Legacy Projects
- Converting Projects Into Directory-Based Format
- Saving Projects as Templates
- Cleaning System Cache
- Switching Between Projects

Creating Projects

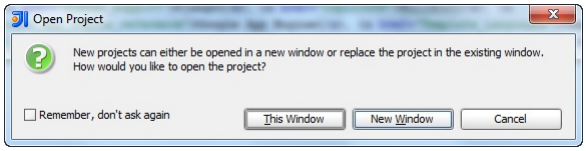
IntelliJ IDEA suggests several ways of creating a project . You can create a project from scratch, or create a project by importing existing source files. These may be the sources originating from Eclipse , Flash Builder , Gradle , or Maven , or a collection of sources of "unspecified origin".

- Creating a Project from Scratch
- Creating a Project by Importing Existing Sources
- Generating a Project from a Framework Template
- Creating a Project Using Yeoman Generator

Creating a Project from Scratch

1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen . Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.
2. On the first page of the wizard , in the left-hand pane, select the project category. This may be the technology that you are going to use, the platform or runtime that your development is going to target, etc.
   The settings in the right-hand part of the page as well as the next steps depend on the selected project category. For corresponding instructions, see:
   - Creating a Java Module
   - Creating a Project for Plugin Development
   - Creating a Maven Module
   - Creating a project with a J2ME module
   - Creating a Flash module
   - Creating a Grails Application Module
   - Creating a Gradle Project
   - Creating a Griffon Application Module
   - Getting Started with Groovy
   - Generating a Project from a Framework Template

Follow the steps of the wizard. When ready, specify whether you want the new project to be opened in a separate window or close the current project and reuse the existing window.



Creating a Project by Importing Existing Sources

You can create a project by importing a collection of sources available on your computer. Such a collection, in addition to source code files, may also include library, configuration, image and other files.

You can import the sources that come from Bnd/Bndtools , Eclipse , Flash Builder , Gradle , or Maven . In such cases the corresponding import operations are referred to as importing from an external model.

You can also import the sources of "unspecified origin" (i.e. the sources that come from the systems other than Eclipse, Flash Builder, Gradle, or Maven). In such cases the sources are treated generically and are referred to simply as existing sources.

In technical terms, importing is just creating IntelliJ IDEA project and module definition files in corresponding locations. (IntelliJ IDEA will create a project with one or more modules .) In most of the cases, the "original" sources will stay where they are.

- Creating a project by importing the sources from Bnd/Bndtools, Eclipse, Flash Builder, Gradle, or Maven
- Creating projects by importing the sources of unspecified origin

Creating a project by importing the sources from Bnd/Bndtools, Eclipse, Flash Builder, Gradle, or Maven

1. If no project is currently open in IntelliJ IDEA, click Import Project on the Welcome screen . Otherwise, select File | New | Project from Existing Sources.
2. In the dialog that opens , select the directory or file which you want to use as a source of import. This may be a directory that contains the corresponding collection of sources or a file that appropriately describes such a collection. (The possible choices are listed right in the dialog.) Click OK.
   As a result, the Import Project wizard opens.
3. On the first page of the wizard , specify the origin of the sources. (This page is not shown if IntelliJ IDEA has guessed what you are importing.) Select Import project from external model, and select Bnd/Bndtools, Eclipse, Flash Builder, Gradle, or Maven. Click Next.

The next steps depend on the origin of the sources that you are importing. For corresponding instructions, see:

- Importing a Project from Bnd/Bndtools Model
- Importing Eclipse Project to IntelliJ IDEA
- Importing Adobe Flash Builder Projects
- Importing Project from Gradle Model
- Importing Project from Maven Model

Creating projects by importing the sources of unspecified origin

1. If no project is currently open in IntelliJ IDEA, click Import Project on the Welcome screen . Otherwise, select File | New | Project from Existing Sources.
2. In the dialog that opens , select the directory which contains the collection of sources you want to import. Click OK.
   As a result, the Import Project wizard opens.
3. On the first page of the wizard , select Create project from existing sources and click Next.
4. On the next page of the wizard , specify the name and location of the project to be created. Note that the folder specified in the Project location field should contain the sources that you are importing. IntelliJ IDEA will try to sort out the files contained in this folder. Click Next.
5. The next page of the wizard shows the directories in which source files are found. Potentially, these directories may be your source root directories .
   Select the directories which you want to include in your project as the source root directories and click Next.
6. On the next page of the wizard , select and configure the libraries to be included in your project. Click Next.
7. On the next page of the wizard , select and configure the modules to be included in your project. Click Next.
8. On the next page of the wizard , specify the SDK to be used in your project and click Next.
9. The next page of the wizard shows the frameworks and technologies detected in your sources. Select the ones that you want to be enabled and click Finish.

Generating a Project from a Framework Template

On this page:

- Overview
- Generating a HTML5 Boilerplate, Web Starter Kit, React Starter Kit, Twitter Bootstrap, or Foundation application stub
- Generating a Node.js Express application stub
- Generating a Dart application stub
- Generating a Meteor application stub
- Generating a PhoneGap/Cordova/Ionic application stub
- Generating an AngularJS application stub
- Generating a Composer project stub
- Generating a WordPress application stub
- Generating a Drupal module stub

Overview

During project creation, IntelliJ IDEA can generate a project or module stub for developing Web applications. The project or module structure is set up and some sources are generated based on external templates and frameworks downloaded upon your request.

IntelliJ IDEA generates project or module stubs based on the following templates:

- Composer project.
- Drupal module.
- WordPress project.
- HTML5 Boilerplate , Twitter Bootstrap , and Foundation for client-side application stubs.
- Node.js Express for the server-side applications using Node.js . and Node.js Boilerplate
- Dart Web application stubs for developing the client side of applications.
- Meteor project stubs for both the client and the server sides of applications.
- PhoneGap / Apache Cordova / Ionic project stubs.
- AngularJS project stubs.

Generating a HTML5 Boilerplate, Web Starter Kit, React Starter Kit, Twitter Bootstrap, or Foundation application stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose the template to use depending on your preferences and the functionality to be implemented in your application:
   - Static Web:
   - HTML5 Boilerplate: choose this option to use the HTML5 Boilerplate template. This is a starting project template that can be easily adapted to your needs.
   - Web Starter Kit: choose this option to use the Web Starter Kit .
   - React Starter Kit: choose this option to use the React Starter Kit .
   - Twitter Bootstrap: choose this option to use the Twitter Bootstrap template, which is an elaborate modular toolkit with rich HTML, CSS, and JavaScript support.
   - Foundation: choose this option to use the Foundation framework.

   Click Next, when ready.
4. On the second page of the wizard, specify the following:
   1. The name of the project and the folder to create it in.
   2. The version of the template to use.

   Click Finish when ready.

Generating a Node.js Express application stub

Generating such application stubs requires that Node.js is supported in IntelliJ IDEA:

1. The Node.js runtime environment is downloaded and installed on your computer.
2. The Node.js repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Learn more in Node.js

To generate a Node.js application stub:

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Node.js and NPM in the left-hand pane.
3. In the right-hand pane, choose Node.js Express App and click Next.
4. On the second page of the wizard, specify the following:
   1. The project name and the folder to create it in.
   2. The path to the Node.js executable file `node.exe` and to the Node.js package manager file `npm.cmd` .
   3. The Express template engine to use. From the Template engine drop-down list, choose one of the following:
      - Jade - haml.js successor.
      - EJS - embedded JavaScript.
      - Hogan.js .
      - Handlebars .
5. Click Finish, when ready. IntelliJ IDEA launches the Express Project Generator tool that downloads all the required data, sets up the project structure, and opens the project either in the current window or in a new one, depending on your choice.

Generating a Dart application stub

Generating Dart application stubs requires that Dart is supported in IntelliJ IDEA:

1. The Dart SDK is available on your computer.
2. The Dart plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

To generate a Dart application stub:

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Dart in the left-hand pane.
3. In the right-hand pane, choose the type of the stub depending on the type of the application you are going to develop. The number of available options depend on the currently used Stagehand, see http://stagehand.pub/ for details. Click Next.
4. On the second page of the wizard, specify the paths to the Dart SDK and to the Dartium executable file. IntelliJ IDEA detects and displays the Dart version. If the SDK was downloaded within the Dart Editor Bundle, the path to Dartium is also detected and filled in automatically. Click Finish when ready.

If necessary, you can open the Dart page of the Settings dialog box by clicking the Settings button and configure command-line options and a custom user data directory for Dartium.

Generating a Meteor application stub

When you click Finish, IntelliJ IDEA generates a skeleton of a Meteor application, including an HTML file, a JavaScript file, a CSS file, and a `.meteor` folder with subfolders. The `.meteor/local` folder, which is intended for storing the built application, is automatically marked as excluded and is not involved in indexing.

By default, excluded files are shown in the project tree. To have the `.meteor/local` folder hidden, click the ⚙ button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.

IntelliJ IDEA also automatically attaches the predefined Meteor library to the project, thus enabling syntax highlighting, resolving references, and code completion. See Configuring JavaScript Libraries for details.

Meteor uses Spacebars templates that are an extension of the Handlebars/Mustache templates. IntelliJ IDEA recognizes Spacebars templates, but as a side effect marks HTML files in Meteor projects with the Handlebars/Mustache icon.

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose Meteor JS App and click Next.
4. On the second page of the wizard, specify the project name and the folder to create it in. Specify the location of the Meteor executable file (see Installing Meteor).
5. From the Template drop-down list, choose the sample to generate. To have a basic project structure generated, choose the Default option.
6. In the Filename text box, type the name for the mutually related `.js` , `.html` , and `.css` files that will be generated. The text box is available only if the Default sample type is selected from the Template drop-dow list.

Generating a PhoneGap/Cordova/Ionic application stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose PhoneGap/Cordova App and click Next.
4. On the second page of the wizard, specify the project name and the folder to create it in. Specify the location of the executable file `phonegap.cmd` , or `cordova.cmd` , or `ionic.cmd` (see Installing PhoneGap/Cordova/Ionic).

When you click Finish, IntelliJ IDEA generates a skeleton of a PhoneGap/Cordova/Ionic application with the framework-specific structure.

Generating an AngularJS application stub

This requires that the AngularJS plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

To generate an AngularJS application stub:

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose AngularJS and click Next.
4. On the second page of the wizard, specify the project name and the folder to create it in.
   From the Version drop-down list, choose the branch https://github.com/angular/angular.js to download the project template from. By default, `master` if shown.
5. When you click Finish , IntelliJ IDEA sets up the AngularJS-specific project structure and generates the `bower.json` and `package.json` files with the definitions of all the necessary dependencies.
6. Download the AngularJs dependencies that contain the AngularJS code and the tools that support development and testing.
   1. Launch the embedded Terminal by hovering your mouse pointer over ▣ in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
   2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
   3. At the command line prompt, type the following command:

```
npm install
```

   Learn more about the installation of dependencies in the Install Dependencies section of the `readme.md` file.

When you click Finish, IntelliJ IDEA generates a skeleton of a AngularJS application with the framework-specific structure.

Generating a Composer project stub

You can have IntelliJ IDEA generate a project stub from an existing package using Composer. In this case, Composer will be initialized and one package will be installed during the project stub creation. After that you can use Composer from the command line or through the user interface.

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. The Project Category and Options dialog opens.
2. In the left-hand pane, choose PHP from the list, then choose Composer Project in the right-hand pane, and then click Next.
3. On the second page of the wizard , specify the project name and the folder where it will be created and provide the following Composer-specific information:
   1. Appoint the Composer instance to use.
      - To use commands from a previously downloaded `composer.phar` , choose Use existing composer.phar: and specify it location in the text box.
      - To have a new instance of Composer downloaded, choose Download composer.phar from getcomposer.org:. The `composer.phar` file will be saved under the project root folder specified in the Location text box.
   2. In the Package area, specify the package to install during the project creation. Select the package to add from the Available Packages list, possibly using the search field, and choose the relevant version from the Version to install drop-down list.
   3. Expand the Settings area and specify the path to the PHP interpreter to use (see Configuring Remote PHP Interpreters and Configuring Local PHP Interpreters for details). In the Command line parameters text box, type the additional command line parameters to adjust the add dependency procedure. For example, to have a package added to the `require-dev` section instead of the default `require` section type `-dev` . For more information about Composer command line options during installation, see https://getcomposer.org/doc/03-cli.md .

When you click Finish, the `create-project` command is invoked with the selected package. This results in creating a Composer project whose configuration and structure depends on the selected package, see https://getcomposer.org/doc/03-cli.md#create-project for details. After that a IntelliJ IDEA project opens.

Generating a WordPress application stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. The Project Category and Options dialog opens.
2. In the left-hand pane, choose PHP from the list, then choose WordPress Plugin in the right-hand pane, and then click Next.
3. On the second page of the wizard, specify the project name and the folder where it will be created. In the WordPress Installation Path text box, specify the folder where WordPress is installed. This folder should contain the `wp-admin` and `wp-includes` subdirectories. Click Finish to start the project stub generation.
4. If the newly created project is outside the WordPress installation folder, configure it as an external library by adding it to the list of included path.
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. On the PHP page that opens, add the path to the WordPress installation folder to the Include Paths list: click the + button and specify the path to the installation folder in the dialog box that opens. Learn more in Configuring Include Paths .

Generating a Drupal module stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. The Project Category and Options dialog opens.
2. In the left-hand pane, choose PHP from the list, then choose Drupal Module in the right-hand pane, and then click Next.
3. On the second page of the wizard , specify the project name and the folder where it will be created and provide the following Drupal-specific information:
   1. Specify the root folder of the Drupal installation in the Drupal installation path. Type the path manually or click the Browse button ... and select the relevant folder in the dialog box that opens.
   2. Select the Set up PHP | Include paths check box to have Drupal include paths automatically configured for the project. The following paths will be added to the Include Paths list on the PHP page of the Settings dialog box:
      1. `<drupal installation root>/includes`
      2. `<drupal installation root>/modules`
      3. `<drupal installation root>/sites/all/modules`

      The include paths are added to the list after you leave the dialog box.
   3. From the Version drop-down list, choose the version of Drupal to use, the supported versions are 6, 7, and 8 .

You can update integration settings at any time later in the File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.

IntelliJ IDEA generates a project stub in accordance with the selected Drupal version. For Drupal 8, a `module_name.info.yml` file is generated where information about the module is provided.

Creating a Project Using Yeoman Generator

IntelliJ IDEA supports integration with the Yeoman tool and provides interface for using it in generating framework-specific project stubs.

On this page:

- Before You Start
- Installing Yeoman
- Configuring a List of Project Stub Generators
- Creating a Project by a Generator

Before You Start

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The Yeoman tool is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the Yeoman tool.
2. If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Yeoman tool and npm from any folder.
3. Make sure the Yeoman and the NodeJS plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Installing Yeoman

The easiest way to install Yeoman is to use the Node Package Manager (npm), which is a part of Node.js . Yeoman can be installed both globally or locally, in a project. It is recommended that you install globally because in this case you can run it from any folder but not from its installation folder.

IntelliJ IDEA provides interface for global and local installation modes. Alternatively, you can install Yeoman manually through the command line.

To install Yeoman from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the yo package from the list.
4. For global installation, select the Options check box and type `-g` in the text box next to it.
5. Click Install Package to start installation.

To run the installation from the command line:

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
2. Type the following command at the command line prompt:

```
npm install -g yo
```

   where `-g` means global installation

Configuring a List of Project Stub Generators

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
2. In the right-hand pane, choose Yeoman and click Next.
3. The second page of the wizard shows all the previously installed Yeoman generators . Click Install Generator.
4. From the dialog box that opens showing all the available generator packages, select the required package in the left-hand pane and click the Install Generator button that appears in the right-hand pane. You can install several packages one after another without leaving the dialog box.
   When the installation is over, click Close to return to the list of generators which is already expanded with the newly added package.

Creating a Project by a Generator

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
2. In the right-hand pane, choose Yeoman and click Next.
3. The second page of the wizard shows all the previously installed Yeoman generators .
   Select the required generator from the list and click Next.
4. Specify the required settings in the New Project wizard that starts. The number of pages and their contents depend on the chosen generator.
5. On the last page of the wizard, select or clear the Run npm install&bower install check box to specify whether you want to run Node Package Manager and Bower to install the packages that are required for developing the new project.
6. Click Next and choose to open the new project in the current window or in the new one.

Opening, Reopening and Closing Projects

On this page:

- Basics
- Opening an existing project
- Reopening a recent project
- Closing a project

Basics

In the Project Opening section of this page, you can choose one of the following options:

- Open project in a new window to open a new IntelliJ IDEA window each time a new project is opened.
- Open project in the same window to stay in the same IntelliJ IDEA window.
- Confirm window to open project in to keep the default behavior and display a dialog box to choose how to open each new project.

Opening an existing project

To open an existing project

1. Do one of the following:
   - On the Welcome screen , click Open.
   - On the main menu, choose File | Open.
2. In the Open Project dialog box, navigate to the desired `*.ipr` , `.classpath` , `.project` , or `pom.xml` , file or to the folder marked with the icon, that contains project `.idea` directory.
3. In the Select Path dialog box, select the directory that contains the desired project.

   Tip Drag the desired project from your file chooser right to the Open Project dialog without locating it there. The respective file in the dialog will be found automatically.
4. Specify whether you want to open the project in a new frame, or close the current project and reuse the existing frame. Refer to the section Opening Multiple Projects for details.

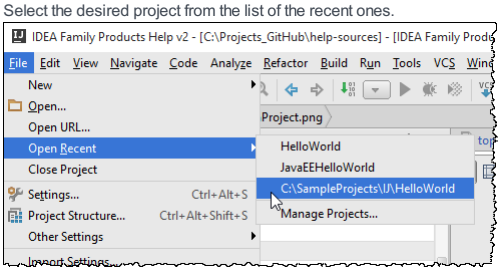
Reopening a recent project

IntelliJ IDEA keeps the history list of the recent projects, from which you can select the desired one.

When IntelliJ IDEA starts, the most recent project reopens by default (unless this option is disabled in System Settings page of the Settings/Preferences dialog).

To reopen a recent project

1. On the main menu, choose File | Open Recent.
2. Select the desired project from the list of the recent ones.



3. Specify whether you want to open the project in a new frame, or reuse the current frame.

You can also:

- Reopen a recent project from the Welcome screen . To do that, click the project of interest in the left-hand pane where the list of your recent projects is shown. Start typing in the Welcome screen to filter the list of recent projects.
- Terminate the project opening process, without waiting when it finishes loading, by clicking Cancel in the progress window.
- Open projects and individual files from the command line , by specifying the project or file name as a command line argument.
- Remove all projects from the list of recent projects File | Open Recent - Clear List.
- Delete a certain recent project on the Welcome screen.

Closing a project

When the only open project is closed, the Welcome screen is displayed. In case of multiple projects, each one is closed with its frame.

To close a project

- On the main menu, choose File | Close Project.

Opening Multiple Projects

On this page:

– Basics
– Prerequisite
– Opening project in a new window

Basics

IntelliJ IDEA allows you to work with several projects simultaneously, each one in a separate window. The projects are independent, and cannot share information, except for the Clipboard operations. All the projects run in the same instance of IntelliJ IDEA and use the same memory space.

Tip This behavior is controlled by the options in the section Project opening in the System Settings page of the Settings dialog.

Prerequisite

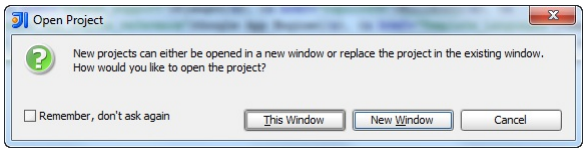
Make sure that one of the options Open project in a new window or Confirm window to open project in in the System Settings page of the Settings dialog is selected.

Opening project in a new window

To open a project in a new window, follow these steps

1. Open a project , while another one is already open.

2. Depending on the selected behavior:
   – If the option Open project in a new window is selected, IntelliJ IDEA will silently open new project in a new window.
   – If the option Confirm window to open project in, IntelliJ IDEA will prompt you to specify whether you want to open project in a new window, or close the current project and reuse the existing window.



Click the New Window button to leave the current project in its window and open the new one in a separate window.

Click This Window button to close the current project and opening the new one in the same window.

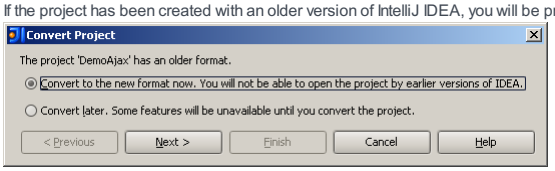
Working With Legacy Projects

When you open a project created in one of the previous versions of IntelliJ IDEA, it is converted to the new format. This action is irreversible. After conversion, the older version of IntelliJ IDEA will not be able to read this project any more. However, IntelliJ IDEA creates backup copies of the project files in the special folders projectFilesBackup<n>.

If a project is not converted to the new format, it can still be used, but some of the IntelliJ IDEA features become unavailable. However, you can convert the project later.

To open a legacy project

1. Open an existing project .
2. If the project has been created with an older version of IntelliJ IDEA, you will be prompted to convert the project immediately, or do it later.



3. Select the desired option and click Finish.

Converting Projects into Directory-Based Format

Any IntelliJ IDEA project created in the file-based format can be easily converted into the directory-based format , which is more convenient when project settings are frequently modified by the team members.

To convert the file-based project format into the directory-based

1. On the main menu, choose File | Save as Directory-Based Format.
   IntelliJ IDEA converts the project configuration files and then reopens the project.

2. Specify whether you want to open the project in a new window .

When the project is opened, all the project settings are stored in the `.xml` files in the `.idea` folder, and it is now safe to delete the `.ipr` and the `.iws` files.

Saving Projects as Templates

On this page:

- Introduction
- Creating a project template
- Deleting a project template

Introduction

You can save any project as a template. If you do so, you'll be able to create template-based projects and modules, that is, projects and modules that are nearly identical to the template project.

The projects that you have saved as templates appear in the User-defined category in the New Project and the New Module wizards.

Unnecessary templates can be deleted in the Manage Project Templates Dialog .

Creating a project template

1. Open the desired project in IntelliJ IDEA.
2. In the main menu, select Tools | Save Project as Template.
3. In the Save Project as Template Dialog that opens:
   - Specify the template name and optionally provide a brief template description.
   - Specify whether the entire project or only one of its modules should be used as a basis for the new template. (This option is available only for projects that contain more than one module.)

Deleting a project template

1. In the main menu, select Tools | Manage Project Templates.
2. In the Manage Project Templates Dialog that opens, select the template or templates you want to delete.
3. Click — or press `Alt+Delete`.

Cleaning System Cache

IntelliJ IDEA caches a great number of files, therefore the system cache may one day become overloaded. In certain situations the caches will never be needed again, for example, if you work with frequent short-term projects. Also, the only way to solve some conflicts is to clean out the cache.

Warning! Cleaning out the system caches, keep in mind that:

- It results in clearing the local history.

  To avoid losing data, check in the changes to your version control system before invalidating caches.
- Causes a complete rebuild of all the projects ever run in the current version of IntelliJ IDEA

Cleaning out the system caches

To clean out the system caches:

- On the main menu, choose File | Invalidate Caches/Restart. The Invalidate Caches message appears informing you that the caches will be invalidated and rebuilt on the next start. Use buttons in the dialog to invalidate caches, restart IntelliJ IDEA or both.

It is important to note the following:

- The files are not actually deleted until IntelliJ IDEA restarts.
- Opening and closing a project does not result in deleting any files.

Switching Between Projects

If you are working with several projects opened in different IntelliJ IDEA windows, you can switch between them. The active project is brought forward and gets the focus.

To switch between the currently opened projects, do one of the following

- On the main menu, choose Window | Next Project Window/Previous Project Window.
- Press `Ctrl+Alt+Close Bracket` or `Ctrl+Alt+Open Bracket`.
- On the main menu, point to Window, and then select the name of the desired project.

Creating and Managing Modules

In this section:

- Adding Modules to a Project
- Grouping Modules
- Removing Modules from a Project

Adding Modules to a Project

When necessary, you can add modules to your project . When doing so, you can create new modules from scratch, or by importing existing sources. These may be the sources originating from Eclipse , Flash Builder , Gradle , or Maven , or a collection of sources of "unspecified origin".

You can also add existing modules to a project.

- Creating a Module from Scratch
- Creating a Module by Importing Existing Sources
- Adding an Existing Module to a Project
- Creating a Java Module

Creating a Module from Scratch

On this page:

- Creating a new module from scratch
- Opening the New Module wizard

Creating a new module from scratch

1. Open the project you want to add a module to, and select File | New | Module.
   As a result, the New Module wizard opens. (There are also other ways of accessing this wizard, see Opening the New Module wizard.)
2. On the first page of the wizard , in the left-hand pane, select the module category. This may be the technology that you are going to use, the platform or runtime that your development is going to target, etc.
   The settings in the right-hand part of the page as well as the next steps depend on the selected module category. For corresponding instructions, see:
   - Creating a Java Module
   - Creating a Project for Plugin Development
   - Creating a Maven Module
   - Creating a project with a J2ME module
   - Creating a Flash module
   - Creating a Grails Application Module
   - Creating a Gradle Project
   - Creating a Griffon Application Module
   - Getting Started with Groovy
   - Generating a Project from a Framework Template

Opening the New Module wizard

Provided that a project is open in IntelliJ IDEA, you can access the New Module wizard from:

- The main menu: File | New | Module.
- The Project tool window: right-click a module folder and select New | Module.
- The Project Structure dialog: in the leftmost pane select Modules, above the pane to the right click + and select New Module.

Creating a Module by Importing Existing Sources

You can create a module (or a number of modules) by importing a collection of sources available on your computer. Such a collection, in addition to source code files, may also include library, configuration, image and other files.

You can import the sources that come from Bnd/Bndtools , Eclipse , Flash Builder , Gradle , or Maven . In such cases the corresponding import operations are referred to as importing from an external model.

You can also import the sources of "unspecified origin" (i.e. the sources that come from the systems other than Eclipse, Flash Builder, Gradle, or Maven). In such cases the sources are treated generically and are referred to simply as existing sources.

In technical terms, importing is just creating a module .iml file and adding the module definition to a project file . (If a number of modules is created, one `.iml` file is created for each of the modules.) In most of the cases, the "original" sources will stay where they are.

- Creating a module by importing the sources from Eclipse, Flash Builder, Gradle, or Maven
- Creating a module by importing the sources of unspecified origin
- Opening the Import Module wizard

Creating a module by importing the sources from Eclipse, Flash Builder, Gradle, or Maven

1. Open the project into which you want to import the sources, and select File | New | Module from Existing Sources.
2. In the dialog that opens , select the directory or file which you want to use as a source of import. This may be a directory that contains the corresponding collection of sources or a file that appropriately describes such a collection. (The possible choices are listed right in the dialog.) Click OK.
   As a result, the Import Module wizard opens. (There is also another way of accessing this wizard, see Opening the Import Module wizard.)
3. On the first page of the wizard , specify the origin of the sources. (This page is not shown if IntelliJ IDEA has guessed what you are importing.) Select Import module from external model, and select Bnd/Bndtools, Eclipse, Flash Builder, Gradle, or Maven. Click Next.

The next steps depend on the origin of the sources that you are importing. For corresponding instructions, see:

- Importing a Project from Bnd/Bndtools Model
- Importing Eclipse Project to IntelliJ IDEA
- Importing Adobe Flash Builder Projects
- Importing Project from Gradle Model
- Importing Project from Maven Model

Creating a module by importing the sources of unspecified origin

1. Open the project into which you want to import the sources, and select File | New | Module from Existing Sources.
2. In the dialog that opens , select the directory which contains the collection of sources you want to import. Click OK.
   As a result, the Import Module wizard opens. (There is also another way of accessing this wizard, see Opening the Import Module wizard.)
3. On the first page of the wizard , select Create module from existing sources and click Next.
4. The next page of the wizard shows the directories in which source files are found. Potentially, these directories may be your source root directories .
   Select the directories which you want to include in your project as the source root directories and click Next.
5. On the next page of the wizard , select and configure the libraries to be included in your project. Click Next.
6. On the next page of the wizard , select and configure the modules to be included in your project. Click Next.
7. The next page of the wizard shows the frameworks and technologies detected in your sources. Select the ones that you want to be enabled and click Finish.

Opening the Import Module wizard

Provided that a project is open in IntelliJ IDEA, you can access the Import Module wizard from:

- The main menu: File | New | Module from Existing Sources.
- The Project Structure dialog: in the leftmost pane select Modules, above the pane to the right click + and select Import Module.

Adding an Existing Module to a Project

You can add existing modules to a project. Normally, you do that to add a module from a different project.

To add an existing module to a project

1. Open the project you want to add a module to, and select File | New | Module from Existing Sources.
2. In the dialog that opens , select the module of interest (represented by the corresponding .iml module file), and click OK.

Creating a Java Module

Java modules let you develop Java SE and Java EE applications. The core functionality provided by a Java module can be extended by adding support for various technologies and frameworks.

You can create a Java module when creating a project. You can also add a Java module to an existing project.

To create a Java module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Java .

3. In the right-hand part of the page, specify the SDK (JDK) to be used. Select the JDK from the list, or click New, select JDK and select the installation folder of the desired JDK.
   If necessary, select additional technologies to be supported and specify the associated settings. For more information, see Additional Libraries and Frameworks .

   Click Next.

4. Sometimes, you have an option of creating a template-based sample project. In such cases, the page is shown with the Create project from template check box.
   If you select this check box, a sample Java class will be created in your project along with a run configuration for that class.

5. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Grouping Modules

If there are too many modules in your project, you may want to arrange them in module groups.

The module groups can be nested, that is, a group can contain subgroups, the subgroups - lower-level subgroups, etc.

To organize modules in groups

1. In the Project tool window or in the Project Structure dialog , right-click the desired module.
2. In the context menu, choose Move Module to Group.
3. In the submenu, choose one of the following:
   - Outside any group: If the selected module is currently a member of a module group, the module will be removed from the group. Otherwise, nothing will happen.
   - New top-level group: A new (top-level) module group will be created, and the selected module will become its member.
   - To this group: The selected module will be moved to the selected module group.
   - To new sub group: A new subgroup will be created in the selected group, and the selected module will be moved to this subgroup.

Removing Modules from a Project

On this page:

- Introduction
- Removing a module in the Project tool window
- Removing a module in the Project Structure dialog

Introduction

You can remove unnecessary modules from your project. However, the files and folders that make up the corresponding modules are not physically deleted. Thus, the modules you have removed from a project can later be added to the same or a different project.

To remove a module, you can use the Project tool window or the Project Structure dialog .

Removing a module in the Project tool window

To remove a module in the Project tool window

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Select the module to be removed.
3. Do one of the following:
   - Select Remove Module from the context menu.
   - Select Edit | Remove Module.
   - Press `Delete`.

Removing a module in the Project Structure dialog

To remove a module in the Project Structure dialog

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, under Project Settings, select Modules.
3. In the pane to the right, select the module to be removed.
4. Do one of the following:
   - Click —.
   - Select Delete from the context menu.
   - Press `Delete`.
5. Click OK in the Project Structure dialog.

Configuring Project and IDE Settings

In this section:

- Configuring Project and IDE Settings
  - Introduction
- Accessing Default Settings
- Accessing Settings
- Configuring Code Style
- Configuring Colors and Fonts
- Configuring Color Scheme for Consoles
- Configuring Keyboard Shortcuts
- Configuring Line Separators
- Configuring Menus and Toolbars
- Configuring Quick Lists
- Copying Code Style Settings
- Creating and Registering File Types
- Configuring Scopes and File Colors
- Configuring Third-Party Tools
- Configuring Individual File Encoding
- Switching Between Schemes
- Switching Boot JDK
- Sharing Your IDE Settings
- Exporting and Importing Settings

Note Find descriptions of the other specific configuration procedures in the relevant procedural parts.

Introduction

Project settings refer to a set of preferences related to resources, file colors, version control options, code styles, etc. Project settings are stored with each specific project as a set of `xml` files under the `.idea` folder.

You can configure project settings on the two possible levels:

- The level of a template project. The settings defined for a template project, apply to any project you create.
- The project level. The settings defined on this level apply to the current project only.

Accessing Default Settings

IntelliJ IDEA helps define settings and structure of a default project. These settings are used as defaults every time you create a new project.

On this page:

– Accessing default project settings

– Accessing the default project structure

To access default project settings

1. On the main menu, choose File | Other Settings | Default Settings.

2. Define the desired settings in the Settings dialog box that opens.

To access the default project structure

– On the main menu, choose File | Other Settings | Default Project Structure.

– Define the desired settings in the Project Structure dialog box that opens.

Accessing Settings

On this page:

– Opening the Settings / Preferences dialog

– Finding an option or setting

– Finding an option or setting using Search Everywhere or Find Action

Opening the Settings / Preferences dialog

Do one of the following:

– Press `Ctrl+Alt+S`.

– On the main toolbar, click .

– On the main menu, choose File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X

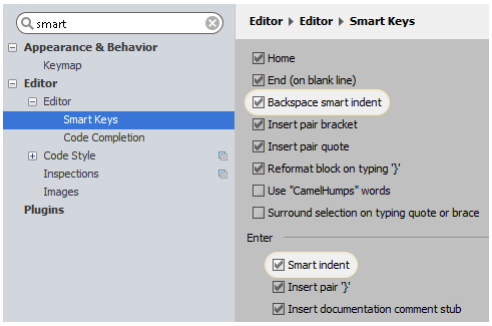
– Press `Ctrl+Shift+A`, type `settings` and press `Enter`. See Navigating to Action.

– Click in the upper-right corner of the IntelliJ IDEA window, and type `#`.

Refer to Finding an option or setting using Search Everywhere or Find Action below.

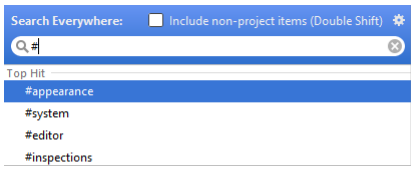
Finding an option or setting

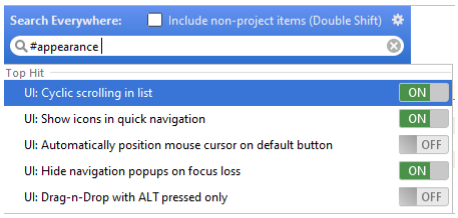
1. Open the Settings / Preferences dialog.
2. In the search field, start typing the text that you expect to find in the name of the setting. As soon as the specified text is found, the matching element is highlighted and the corresponding page is displayed.

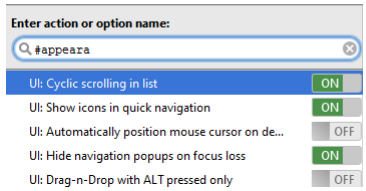


Finding an option or setting using Search Everywhere or Find Action

You can also use Searching Everywhere or Find Action. To find an option or setting, first type `#` character, and then choose one of the suggested categories:







Configuring Code Style

On this page:

- Basics and definitions
- Configuring code style for a language
- Copying code style settings from other languages
- Applying framework-specific pre-configured coding standards
- Configuring the code style for a project using EditorConfig

Basics and definitions

If certain coding guidelines exist in a company, one has to follow these guidelines when creating source code. IntelliJ IDEA helps maintain the required code style.

The code styles are defined at the project level and at the global level.

- At the Project level, settings are grouped under the Project scheme, which is predefined and is marked in bold font. The Project style scheme applies to the current project only.
  The default scheme cannot be changed! When any modifications are made, IntelliJ IDEA automatically creates a copy of the default scheme. If you want to use a different project level scheme, choose the desired scheme and copy its settings to project.
- At the Global level, settings are grouped under the predefined Default scheme, and any other scheme created by the user. The global settings are used when the user doesn't want to keep code style settings with the project and share them.

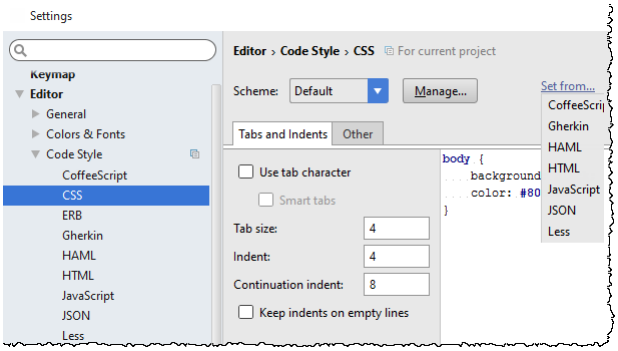
Configuring code style for a language

1. In Settings, click Code Style , and then click the language in question.
2. Choose the code style scheme to be used as the base for your custom coding style for the selected language.
3. Browse through the tabs of the selected language page, and configure code style preferences for it.

Copying code style settings from other languages

For most of the supported languages, you can copy code style settings from other languages or frameworks.

1. In Settings, click Code Style , and then click the language in question.
2. Click the link Set From in the upper-right corner. This link appears for those languages only, where defining settings on the base of the other languages is applicable.
3. In the drop-down list that appears, click the language to copy the code style from:



Applying framework-specific pre-configured coding standards

For PHP files, you can have framework-specific pre-configured coding standards applied.

1. In Settings, click Code Style , and then click the language in question.
2. Click the Set from link, choose Predefined, then choose the relevant pre-configured standard.

Configuring the code style for a project using EditorConfig

Before you start working with EditorConfig, make sure that the EditorConfig plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Make sure that the check box Enable EditorConfig Support is selected in Editor | Code Style.

For more information, see EditorConfig Website.

To configure the code style for a project using EditorConfig:

1. In the project tree, right-click a directory where you want to create the EditorConfig settings file and select New | File.
2. In the dialog that opens, enter `.editorconfig` and click OK.
   IntelliJ IDEA creates an EditorConfig settings file and displays a notification in the pop-up window.

   Every time you open a file, the EditorConfig plugin looks for a file named `.editorconfig` in the directory of the opened file and in every parent directory. A search for `.editorconfig` files will stop if the root file path is reached or an EditorConfig file with `root = true` is found. Therefore, if you want to use the IDE settings instead of the EditorConfig settings, clear the Enable EditConfig Support check box in Editor | Code Style that is selected by default.
3. Start defining your code style settings. Save(Ctrl+S) your file. Every time you modify the `.editorconfig` file, save the file to apply changes to your project.

   The EditorConfig code style configuration overrides the code style configuration in the IDE settings.

Configuring Colors and Fonts

On this page:

– Basics

– Configuring colors and fonts

– Example

Basics

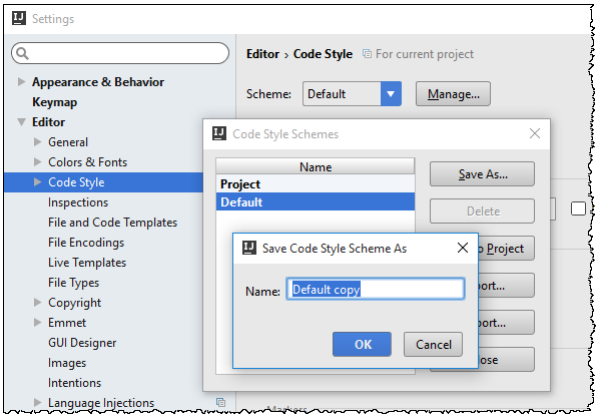
With IntelliJ IDEA, you can maintain your preferable colors and fonts layout for syntax and error highlighting in the editor, search results, Debugger and consoles via font and color schemes.

IntelliJ IDEA comes with a number of pre-defined color schemes. You can select one of them, or create your own one, and configure its settings to your taste. Note that pre-defined schemes are not editable. You have to create a copy of a scheme, and then change it as required.

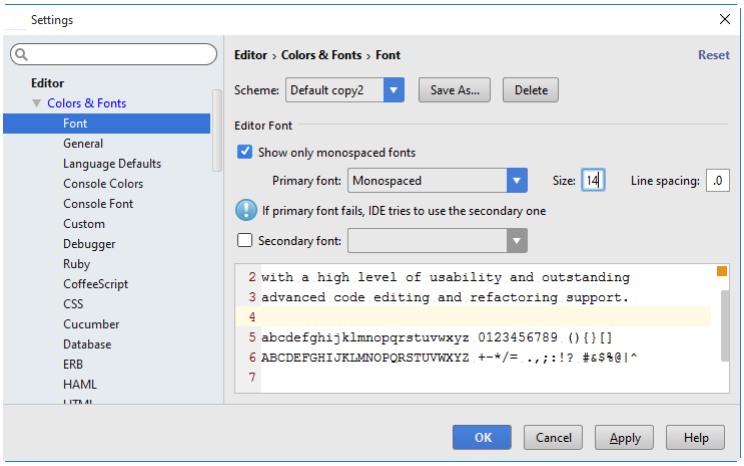
Configuring colors and fonts

To configure color and font scheme

1. Open Settings, and under the Editor node, click Colors & Fonts .
2. Select the desired scheme from the Scheme name drop-down list.
3. If you need to change certain settings of the selected scheme, create its copy. To do that, click the Manage, then Save as button, and type the new scheme name.



4. Under the Colors and Fonts node, change the font families used in the editor and in the console:
   – Define font family for the editor and console. When you just open the Font or Console Fonts pages under the Colors and Fonts node, IntelliJ IDEA displays Editor Font area where you can configure primary and secondary fonts, their size and line spacing.



5. Under the Colors and Fonts node, open pages to configure specific color preferences and font types for the different supported languages and IntelliJ IDEA components.

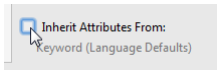
Tip You can view how the new scheme looks in the editor. To do that, just click Apply without closing the Settings dialog box.

Example

Changing font for JavaScript

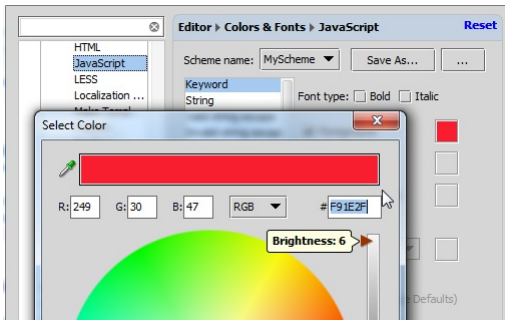
Click JavaScript node.

Important! Make sure the Inherit attributes from <component> check box is cleared:



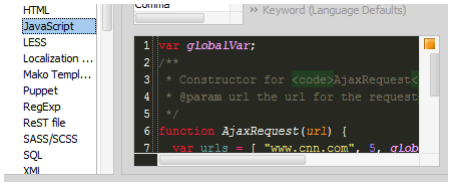
In the list of language components select Keyword.

Define the desired font type using the Bold and Italic check boxes. Next, select the Foreground check box, and right click the color preview to customize:



Refer to the description of the Color Picker dialog box for details.

Observe results in the preview pane:



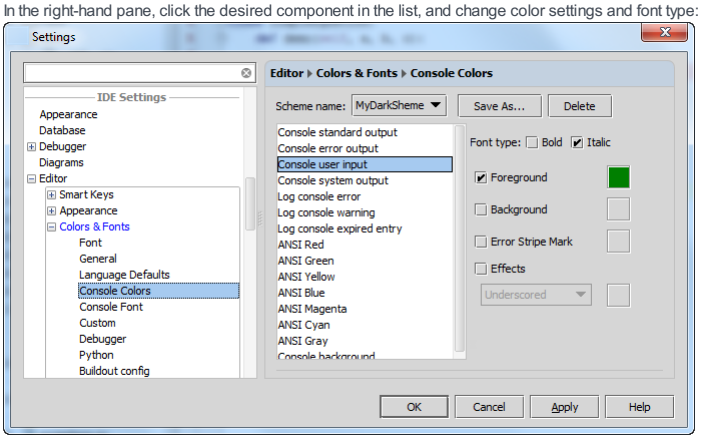
Configuring Color Scheme for Consoles

IntelliJ IDEA enables you to define your habitual color scheme for the various types of consoles. So doing, you can individually configure all sorts of console output and user input.

Color scheme includes numerous colors for background, user input, system output, and error output.

To configure color and font scheme for consoles

1. Make sure you are working with an editable scheme.
2. Open the IDE Settings , and under Colors&Fonts , scroll through the list of components, and select the ones related to consoles:
   - Console Colors
   - Console Fonts
3. In the right-hand pane, click the desired component in the list, and change color settings and font type:



Configuring Keyboard Shortcuts

On this page:

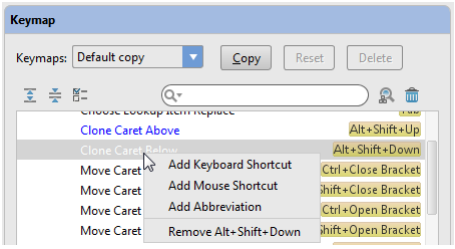
– Basics
– Where the keymaps are stored?
– Configuring keyboard shortcuts and mouse shortcuts
– Searching for actions

Basics

IntelliJ IDEA is a keyboard-centric IDE. Most of the actions (navigation, refactoring, debugging, etc.) can be carried out without using a mouse, which lets dramatically increase coding speed. If you had used another IDE for a while and have memorized your favorite keyboard shortcuts, you can use them all in IntelliJ IDEA.

IntelliJ IDEA completely suits your *shortcut habits* by supporting customizable keymaps. A keymap is a set of keyboard and mouse shortcuts that invoke different actions - menu commands, editor operations, etc. IntelliJ IDEA comes with a set of pre-configured keymaps.

Pre-configured keymaps are not editable. If you need to change some shortcuts, the copy of the current selected pre-defined keymap is created automatically:



Where the keymaps are stored?

All user-defined keymaps are stored in separate configuration files under the `config/keymaps` subdirectory in the IntelliJ IDEA profile directory:

– Windows and *NIX systems: `<User home>/.IntelliJ IDEA<xx>/config/keymaps`
– OS X: `~/Library/Preferences/IntelliJ IDEA<xx>/keymaps/`

Each keymap file contains only differences between the current and the parent keymaps.

Configuring keyboard shortcuts and mouse shortcuts

To configure keyboard shortcuts and mouse shortcuts

1. Open the Settings dialog box , and click Keymap .
2. Select one of the pre-configured Keymaps, which you want to use as the base for the new one, and click Copy. Accept the default name, or change it as required.
3. In the content pane of actions, select the desired action.
4. Configure keyboard shortcuts. To do that, follow these steps:
   1. Click on the toolbar, or right-click the selected action, and choose Add Keyboard Shortcut. Enter Keyboard Shortcut dialog box opens.
   2. Press the keys to be used as shortcuts. The keystrokes are immediately reflected in the First Stroke field. Optionally, select the check box next to Second Stroke and press keys to be used as alternative keyboard shortcuts.

      As you press the keys, the Preview field displays the suggested combination of keystrokes, and the Conflicts field displays warnings, if some of the keystrokes are already assigned to the other actions.
   3. Click OK with the mouse pointer to create a shortcut and bind it with an action.

   It is important to use the mouse pointer, because any keystroke is interpreted as a shortcut.
5. Configure mouse shortcuts. To do that, follow these steps:
   1. Click on the toolbar, or right-click the selected action, and choose Add Mouse Shortcut on the context menu, if you need to bind an action to a mouse click. Enter Mouse Shortcut dialog box opens.
   2. In the Click Count section, click a radio button to choose a Single Click or Double Click.
   3. Hover your mouse pointer over the section Click Pad and click the desired mouse button. Use `Alt` , `Ctrl` , and `Shift` modifiers for diversity. As you click, the Shortcut Preview field displays the suggested shortcut, and the Conflicts field displays warnings, if some of the shortcuts are already assigned to the other actions.
   4. Click OK or Press `Enter` to create a shortcut and bind it with an action.

If a conflict is reported, a warning message appears. You can choose one of the following options:

– Remove to remove all other bindings and preserve the new one.
– Leave to preserve all bindings including the new one.
– Cancel to return to the keymap definition.

Although you can ignore conflict and bind a shortcut with several actions, it is strictly recommended to avoid binding two actions with the same shortcut, because the order of performing such actions is not defined.

Searching for actions

Tip Use the Keymap page to search for certain actions by name, or by shortcut.

To find an action by name, type this name in the search field

. As you type, the content pane shows actions with the matching names.

To find an action by shortcut, click . In the Filter Settings dialog box, start pressing keys. The content pane shows only the actions with the matching shortcuts.

Click your mouse somewhere outside the Filter Settings dialog box to close it. Avoid using keys, since any keystroke is interpreted as a shortcut.

Configuring Line Separators

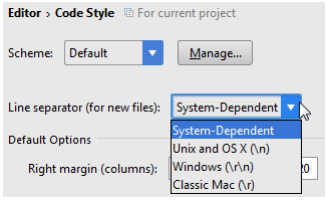
IntelliJ IDEA makes it possible to set up line separators (line endings) for the newly created files, and change line separator style for the existing files.

On this page:

- Setting up line separators for newly created files
- Viewing line ending style for the current file
- Changing line separator for a file, currently opened in the editor
- Changing line separator for a selection in the Project view
- Tips and tricks

To set up line separators for new files

1. In Settings, click Code Style .
2. From the Line separator (for new files) drop-down list, select the desired line separator style:



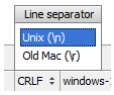
3. Apply changes and close the dialog.

To view line ending style for the current file

1. Open the desired file in the editor, as described in the section Opening and Reopening Files .
2. View the Status bar : the current line endings style is denoted by the dedicated icon with the specified style, for example, CRLF .

To change line separator for a file, currently opened in the editor

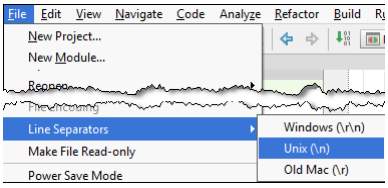
1. Open the desired file in the editor, as described in the section Opening and Reopening Files .
2. Do one of the following:
   - Click the line separator spin box in the Status bar , and choose the desired line ending style from the pop-up menu:



   - Choose File | Line Separators on the main menu, and then choose the desired line ending style from the sub-menu.

To change line separator for a file or directory, selected in the Project view

1. Select a file or directory in the Project tool window .
   Note that if a directory is selected, the line ending style applies to all nested files recursively.
2. Choose File | Line Separatorson the main menu, and then select the desired line ending style from the sub-menu.



Tips and tricks

- Use multiple selection in the Project view.
- Changing line separator is reflected in the Local history of a file.
- Run the inspection 'Inconsistent line separators' to find out, which files use line separator different from project's default.

Configuring Menus and Toolbars

You can customize menu and toolbar command lists to regroup features or make your favorites easier to access.

To customize menus and toolbars

1. Open Settings and click Menus and Toolbars . Alternatively, right-click the main toolbar, and choose Customize Menus and Toolbars on the context menu.

2. In the list of available menus and bars, expand the node you want to customize and select the desired item.

3. Customize the list of items in the selected menu or bar using the buttons on the right from the list:
   - To add a new command, select the desired location in the list and click the Add After button. In the Choose Action To Add dialog box that opens, select the desired action.
     Optionally associate the action with an icon using the Icon Path text box. In this text box, specify the location of the file with the icon you want to assign to the selected action. If necessary, use the Browse button to select the file in the corresponding dialog .

     > Tip The image file should have .png extension.
     > – The size of the toolbar icons should be 16x16.

   - To change the icon associated with a command, select the desired command in the list and click the Edit Action Icon button. In the Choose Actions Icon Path dialog box that opens, specify the location of the desired image. If necessary, use the Browse button to select the image in the Select Path Dialog .
   - To delete an item from the list, select it and click the Remove button.
   - To have logical groups of commands separated from each other by a separator, select the desired location in the list and click the Add Separator button.
   - To change the order in which commands appear in the selected menu or on the selected bar, use the Move Up and Move Down buttons.

4. To abandon the changes and return to the default settings, click the Restore Default button.

Configuring Quick Lists

On this page:

- Introduction
- Using a quick list
- Configuring a quick list

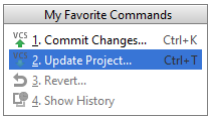
Introduction

A Quick List is a pop-up menu of IntelliJ IDEA commands, configured by the user and associated with a keyboard or mouse shortcut. You can create as many quick lists, as necessary. Each command, included in a quick list, is identified by a sequential number. Numbering starts from the numerals (0 to 9), and then proceeds with the letters in alphabetical order.

Using a quick list

To invoke a command from a quick list

1. Invoke quick list by its keyboard shortcut.
2. Select the desired command, using its number, the mouse cursor, or navigation keys and the `Enter` key:



Configuring a quick list

To configure a quick list

1. Open Settings/Preferences dialog, and click Quick Lists page.
2. Click + to create a new quick list.
3. In the Display name field, specify the name of the quick list. Optionally, provide the quick list description.
4. Configure the quick list. Use:
   - Add to add actions to the list. Select the actions in the Add Actions to Quick List dialog that opens.
   - Add Separator to add a separator at the end of the list.
   - Move Up and Move Down to move the selected item one line up or down in the list.
   - Remove to remove the selected item from the list.
5. Apply the changes.
6. Bind the new quick list with one or more shortcuts:
   - In the Keymaps page of the Settings/Preferences dialog, expand the Quick Lists node and select the new quick list.
   - Perform the key binding procedure . Note that you can only modify a custom keymap.
7. Apply the changes and close the dialog.

Copying Code Style Settings

On this page:

– Introduction

– Creating a global copy of a code style scheme

– Creating a project copy of a code style scheme

Introduction

You can define the code styles that differ from the pre-defined ones. These code style schemes are stored in XML files, in the `config/codestyles` folder under the user home directory.

You can use the created copy for modifying code styles, and for export.

If you select a code style scheme other than Project, then this code style will be saved for a project. Thus you can assign a global code style for each project.

Creating a global copy of a code style scheme

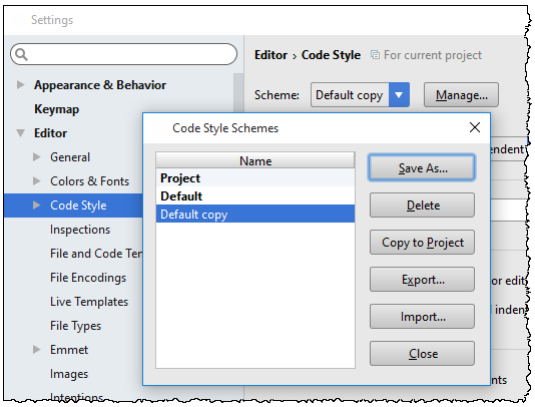
To create a copy of code style settings

1. In the Code Style page , select the desired scheme from the drop-down list, and click Manage.
2. In the Code Style Schemes dialog box that opens, click Save As....
3. In the Save Code Style Scheme As dialog box, type the name of the new scheme.
   IntelliJ IDEA saves the new code style with the specified name in the `config/codestyles/<code_style_name>.xml` file under the IntelliJ IDEA home directory.
4. Close the Code Style Schemes dialog box, and apply changes.

Creating a project copy of a code style scheme

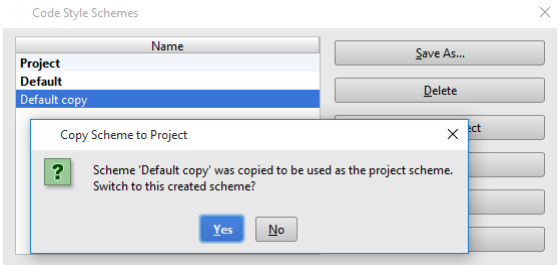
To copy code style settings to a current project

1. In the Code Style page , select the desired scheme from the drop-down list, and click Manage.
2. In the Code Style Schemes dialog box that opens, click Copy to Project:



The selected code style is saved in the `.idea` directory in the file `codeStyleSettings.xml`.

3. In the Copy Scheme to Project dialog box, confirm switching to the new scheme, if necessary (Yes):



Then close the Code Style Schemes dialog box, and apply changes.

Creating and Registering File Types

On this page:

- Introduction
- Creating a file type
- Registering a file type

Introduction

You can create custom file types to enable parsing these files in the editor by defining highlighting schemes for keywords, comments, numbers, etc. To enable IntelliJ IDEA to decide how to treat a file, you need to associate each file type with relevant extensions .

Creating a file type

To create a new file type

1. Open the Settings dialog by choosing File | Settings . Then select . Find more on page Accessing Settings .
2. On the File Types page that opens, click +.
3. In the New File Type dialog box that opens, specify the name of the new type and optionally provide a description.
4. In the Syntax Highlighting section, specify the characters for line and block comments, hex prefixes, and number postfixes.
5. In the Keywords section, specify sets of keywords using the tabs from 1 to 4. To do so, select the desired tab, click + (`Alt+Insert`), and enter the keyword name in the Add New Keyword dialog box that opens.

   Tip Each set of keywords has its own highlighting. You can change the highlighting color scheme for each set, on the Colors and Fonts page. Click the Custom tab and edit the Keyword1, Keyword2, Keyword3, and Keyword4 properties.

Registering a file type

To associate a file type with extensions

1. Open the File Types settings page.
2. From the Recognized File Types list, select the desired type.
3. In the Registered Patterns area, complete the list of patterns that define the file extensions to indicate that the corresponding files belong to the selected type. Do one of the following:
   - To register a new pattern, click + (`Alt+Insert`) and enter the desired extension pattern in the Add Wildcard dialog box that opens.
   - To update a pattern, select it in the list, click the Edit button and make the necessary changes in the Edit Wildcard dialog box that opens.
   - To remove a pattern from the list, select it and click – (`Alt+Delete`).

Configuring Scopes and File Colors

This section describes how to configure scopes and coloring of the files belonging to these scopes:

- Creating a new custom scope
- Configuring the list of items in a custom scope
- Associating file color with a scope
- Arranging the order of scopes

Project scopes are configured in the Scopes page of the Settings dialog box.

To create a new custom scope

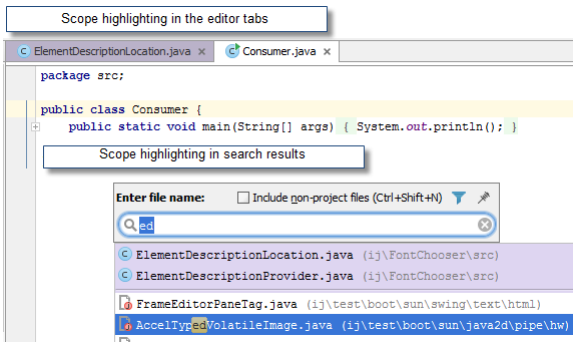
1. In the Scopes settings page, click Add scope.
2. Select Local Changes or Shared from the drop-down list. Shared scopes are defined for the current project and accessible for the team members via VCS, while Local scopes are intended for personal use only and are stored in your workspace.

   Tip You can change the sharing state later using the Share scope check box in the bottom of the page.
3. Specify the name for the new scope.
4. Apply changes.

To configure the list of items in a custom scope

1. In the Scopes settings page, select the scope that you want to configure.
2. Do one of the following:
   - Choose files and folders to be included in the scope and use buttons on the right. Based on the inclusion/exclusion, IntelliJ IDEA creates an expression and displays it in the Pattern field.
   - Specify pattern in the Pattern field manually, using the scope language syntax .
3. Apply changes.

Files belonging to different scopes can be highlighted in different colors throughout the IntelliJ IDEA's user interface: in navigation lists , in the editor tabs, in the Project window . This allows much faster and easier navigation in large projects.



To associate file color with a scope

1. Open the File Colors settings page.
2. Decide whether you want the scope-color association to be only available to you or to be shared with the team. Depending on that, use either Local colors or Shared colors section of the page.

   Tip You can change the sharing state later by selecting a color in the list ans using the Share button in the Local colors section or the Unshare button in the Shared colors section of the page.
3. Click Add.
4. In the dialog box that opens, select a scope and pick a color for it.
5. If necessary, use the check boxes on top of the page to define where in the user interface files belonging to the scopes are highlighted.
6. Apply changes.

If some file is included into several scopes, the order of the scopes becomes important: IntelliJ IDEA uses the color of the uppermost scope (shown in the Scopes settings page) to highlight such file. Of course, you can change the order of the scopes, and thus the resulted highlighting.

To arrange the order of scopes

1. Open the Scopes settings page.
2. Select a scope whose position in the order you want to change.
3. Click Move Up / Move Down
4. Apply the changes.

Configuring Third-Party Tools

You can define third-party standalone applications (code generators and analyzers, pre- and post- processors, database utilities, etc.) as external tools and then run them from IntelliJ IDEA.

You can pass contextual information (like the currently selected file, or your project source path) to the external tools, view the tool output, and more.

The tools are defined on the External Tools page in the Settings dialog and appear as commands in the Tools menu and in various context menus. They can also be assigned keyboard shortcuts (see Configuring Keyboard Shortcuts).

Configuring Individual File Encoding

On this page:

- Basics
- Configuring encoding for a directory or file without embedded encoding information
- Changing encoding of a file with explicit encoding
- Changing encoding of a file without explicit encoding

Basics

There are two modes of dealing with file encoding:

- Converting: the contents of the editor are stored in a different encoding. So doing, the contents of the underlying file change, but the contents of the editor stay unchanged.
- Reloading: the underlying file, opened in the editor, is shown in an encoding that differs from its original one. So doing, the contents of the editor can change, but the underlying files does not.

IntelliJ IDEA suggests the following major ways to change encoding:

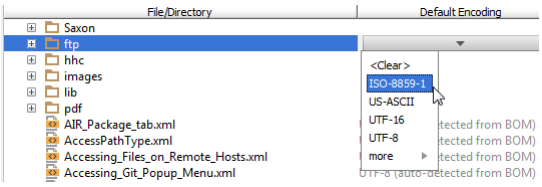
- Using the File Encodings page of the Settings dialog, for directories and for the files that do not contain encoding information.
- Using the Status bar or menu command , for individual files that do not contain encoding information.
- Using the editor , for individual files that contain encoding information.

IntelliJ IDEA also supports configuring encoding for properties files .

Configuring encoding for a directory or file without embedded encoding information

To configure encoding for a directory or file without embedded encoding information

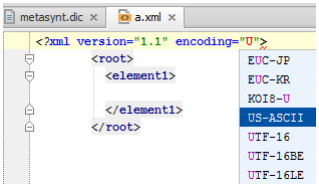
1. In Settings, expand the Editor node and select File Encodings .
2. The File/Directory column shows the tree view of your project. The Default Encoding column shows encoding for directories or files. Click the Default encoding column for a directory or file you want to define encoding for, and then choose the desired encoding from the drop-down list:



Changing encoding of a file with explicit encoding

To change encoding of a file that contains explicit encoding

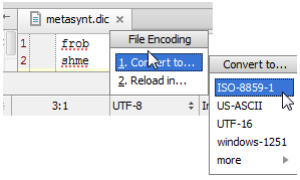
1. Open the desired file in the editor.
2. Change explicit encoding information. Use error highlighting to recognize wrong encoding and press `Ctrl+Space` to have a list of available encodings displayed:



Changing encoding of a file without explicit encoding

To change encoding of a single file that doesn't contain explicit encoding

1. Open the desired file for editing.
2. Do one of the following:
   - On the main menu, point to File | File encoding.
   - Click file encoding on the Status bar .
3. Select the desired encoding from the pop-up window.

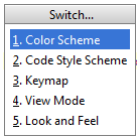


4. If the selected encoding will change the file contents, IntelliJ IDEA shows a dialog box, where you can choose to Reload file from disk, or Convert it to a different encoding.

Switching Between Schemes

You can quickly switch between various color schemes, keyboard layouts, and look-and-feels without actually invoking the corresponding page of the Settings dialog box.

1. Choose View | Quick Switch Scheme on the main menu or press `Ctrl+Back Quote`.
2. In the pop-up window that opens select the desired scheme (Colors and Fonts, Code Style, etc.).



3. In the suggestion list, click the desired option.

Switching Boot JDK

Warning! Regardless of your choice, the selected version of JDK shall be not lower than 1.8.
– Currently, this option is available for OS X and Linux users only

On this page:

– Introduction
– Switching the IDE boot JDK

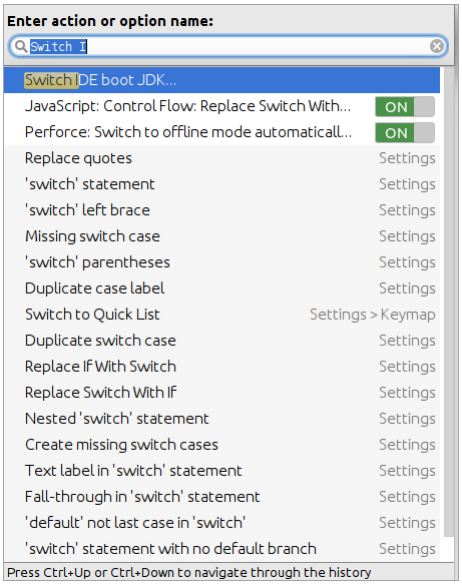
Introduction

In case when you prefer JDK other than bundled with IntelliJ IDEA, you can choose between the latter and another kit, installed on your system.

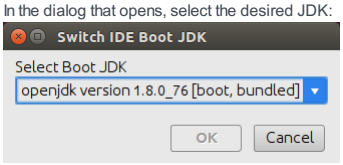
Switching the IDE boot JDK

To switch the IDE boot JDK, do the following:

1. On the main menu, choose Help | Find Action or press `Ctrl+Shift+A`.
2. In the list of actions that appears, find Switch IDE boot JDK action and select it. Simplify your search by typing the first letters:



3. In the dialog that opens, select the desired JDK:



4. Click OK to apply changes.

Sharing Your IDE Settings

On this page:

- Introduction
- Prerequisites
- Configuring settings repository
- Authentication
- Configuring a read-only source

Introduction

IntelliJ IDEA allows you to share your IDE settings between different instances of IntelliJ IDEA (or other IntelliJ platform-based) products installed on different computers.

This is useful if you are using several IntelliJ IDEA installations, or want to implement the same settings among your team members or company-wide.

Prerequisites

Before you start working with Settings Repository, make sure that the Settings Repository plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Configuring settings repository

If you want to share your IDE settings, perform the following steps:

1. Create a repository on GitHub .
2. On the computer where the IntelliJ IDEA instance whose settings you want to share is installed, navigate to File | Settings Repository. Specify the URL of the repository you've created and click Overwrite Remote.
3. On each computer where you want your settings to be applied, in the Settings/Preferences dialog , expand the Tools node and choose Settings Repository , specify the URL of the repository you've created, and click Overwrite Local.
   You can click Merge if you want the repository to keep a combination of the remote settings and your local settings. If any conflicts are detected, a dialog will be displayed where you can resolve these conflicts.

   If you want to overwrite the remote settings with your local settings, click Overwrite Remote.

Your local settings will be automatically synchronized with the settings stored in the repository each time you perform an Update Project or a Push operation, or when you close your project or exit IntelliJ IDEA.

If you want to disable automatic settings synchronization, navigate to File | Settings | Tools | Settings Repository and disable the Auto Sync option. You will be able to update your settings manually by choosing VCS | Sync Settings from the main menu.

Authentication

On the first sync, you will be prompted to specify a username and password.

It is recommended to use an access token for GitHub authentication. If, for some reason, you want to use a username and password instead of an access token, or your Git hosting provider doesn't support it, it is recommended to configure the Git credentials helper .

Note that the OS X Keychain is supported, which means you can share your credentials between all IntelliJ Platform-based products (you will be prompted to grant access if the original IDE is different from the requestor IDE).

Configuring a read-only source

Apart from the Settings Repository, you can configure any number of additional repositories containing any types of settings you want to share, including live templates, file templates, schemes, deployment options, etc.

These repositories are referred to as Read-only sources, as they cannot be overwritten or merged, just used as a source of settings as is.

To configure such repositories, do the following:

1. In the Settings/Preferences dialog , expand the Tools node and choose Settings Repository .
2. Click + and add the URL of the GitHub repository that contains the settings you want to share.

Synchronization with the settings from read-only sources is performed in the same way as for the Settings Repository .

Exporting and Importing Settings

On this page:

– Introduction
– Exporting settings to a JAR archive
– Importing settings from a JAR archive

Introduction

IntelliJ IDEA enables you to preserve and share your working environment. You can archive and store your preferred IDE settings, put the settings file under version control and thus make it available to your colleagues. On the other hand, you can use the settings, defined by the other team members, or your own ones intended for a different usage.

Exporting settings to a JAR archive

To export IDE settings to a JAR archive

1. On the main menu, choose File | Export Settings.
2. In the Export Settings dialog box that opens specify the settings to export by selecting the check boxes next to them. By default, all settings are selected.
3. In the Export settings to text box, specify the fully qualified name of the target archive. Type the path manually or click the Browse button and specify the target file in the dialog that opens .

Importing settings from a JAR archive

To import settings from a JAR archive

1. On the main menu, choose File | Import Settings.
2. In the Import File Location dialog box that opens select the desired archive.
3. In the Select Components to Import dialog box that opens specify the settings to be imported, and click OK. By default, all settings are selected.

Configuring Project Structure

In this part:

- Configuring Project Structure
  - Basics
- Accessing Project Structure
- Accessing Module Settings
- Adding Frameworks Support
- Configuring Content Roots
- Excluding Files from Project
- Configuring Module Dependencies and Libraries
- Configuring Module Compiler Output
- Configuring Project Compiler Output
- Configuring Global, Project and Module SDKs
- Configuring Project and Global Libraries

Basics

For accessing project structure, IntelliJ IDEA provides Project Structure dialog, which can be invoked by the icon on the main toolbar, or using keyboard shortcut `Ctrl+Shift+Alt+S`.

On the project level, you have to define the general settings of a project (project SDK, language level, and the project compiler output), libraries for the current project, facets and artifacts. On the level of each module comprising a project, you have to define the content roots, module compiler output, and module dependencies.

On the global level, you can define the SDKs and libraries that will be available for the other projects.

To access project structure, use the page of the Settings/Preferences dialog, which is invoked by icon on the main toolbar, or keyboard shortcut `Ctrl+Alt+S`.

Accessing Project Structure

This section describes simple steps required to access the project structure.

To access project structure

- Do one of the following:
  - On the main menu, choose File | Project Structure.
  - Press `Ctrl+Shift+Alt+S`.
  - Click on the toolbar.

Accessing Module Settings

On this page:

– Accessing module settings
– Tips and tricks

Accessing module settings

To access module settings

1. Open the Project Structure dialog .
2. In the left-hand pane, under Project Settings, click Modules.
3. In the area under + – , select the module of interest.
   The Module page opens in the right-hand part of the dialog showing the module settings. Configure the settings as required.

Tips and tricks

To access the module settings, you can select the desired module in the Project tool window , Dependency Viewer , or the Module Dependencies tool window , and then do one of the following:

– Choose Open Module Settings in the context menu.
– Press `Ctrl+Enter`.
– Press `F4`.

Adding Frameworks Support

You can extend capabilities of an existing module by adding support for various development frameworks and technologies.

To add frameworks support to a module, do one of the following

- In the Project tool window, right-click the desired module node, and choose Add Framework Support from the context menu. Then, in the Add Frameworks Support Dialog , select the check boxes for the technologies you want to be enabled in your module, and specify the associated settings.
- Open the module settings and click +. Choose the desired facet from the list. Configure the facet settings as required.

Configuring Content Roots

On this page:

- Introduction
- Adding a content root
- Specifying folder categories (sources, tests, etc.)
- Using the Project Structure dialog
- Specifying that the sources or resources are generated
- Using the Project tool window
- Assigning a package prefix to a folder with Java sources
- Specifying the output path for resources
- Removing a content root

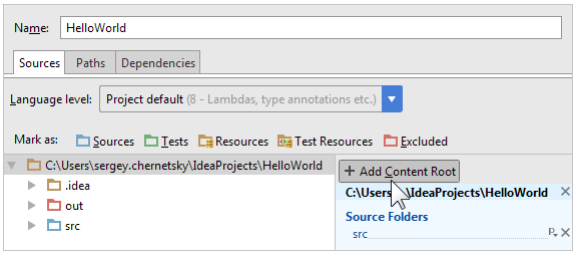
Introduction

When created, a module , normally, has one content root .

You can create additional content roots as well as remove the unnecessary ones.

Adding a content root

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, select Modules.
3. In the pane to the right, select the necessary module.
4. In the right-hand part of the dialog, select the Sources tab.
5. Click Add Content Root.



6. In the dialog that opens , select the folder to be added as a content root, and click OK.
7. Click OK in the Project Structure dialog.

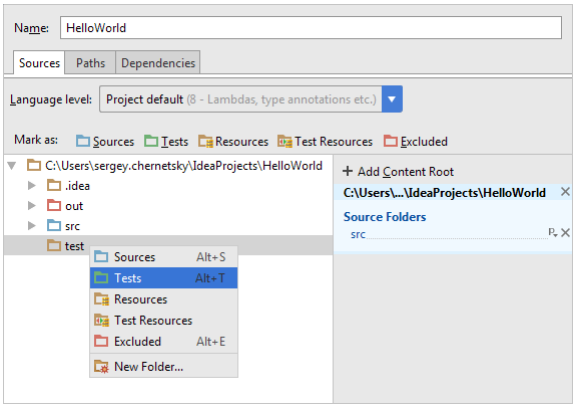
Specifying folder categories (sources, tests, etc.)

Folders within content roots can be assigned to the following categories: sources, test sources, generated sources, generated test sources and excluded folders (in Java modules - also resources and test resources). This can be done in the Project Structure dialog or in the Project tool window.

- Using the Project Structure dialog
- Using the Project tool window

Using the Project Structure dialog

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, select Modules.
3. In the pane to the right, select the necessary module.
4. In the right-hand part of the dialog, select the Sources tab.
5. Select the folder of interest and do one of the following:
   - To the right of Mark as, click the name of the category, e.g. Tests.
   - Select the category (e.g. Tests) from the context menu.



6. Click OK in the Project Structure dialog.

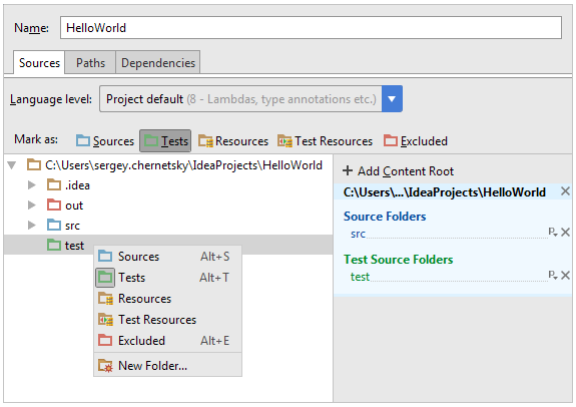
In a similar way you can change the folder category: just select a different category for a folder.

To cancel the folder association with its current category (i.e. to make a folder an "ordinary" folder), do one of the following:

- Click the current category name to the right of Mark as (e.g. Tests if the folder is currently assigned to test sources).

– Click the current category name in the context menu.

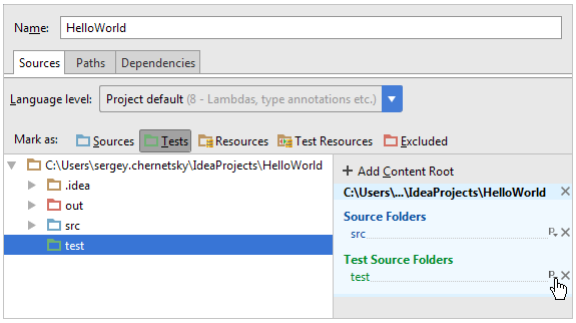
– In the right-hand pane, under the category name (e.g. Test Source Folders), click ✕ to the right of the folder name (or folder path).



Specifying that the sources or resources are generated

To specify that a folder contains generated sources or resources, you should first assign that folder to sources, resources, test sources or test resources, and then do the following:

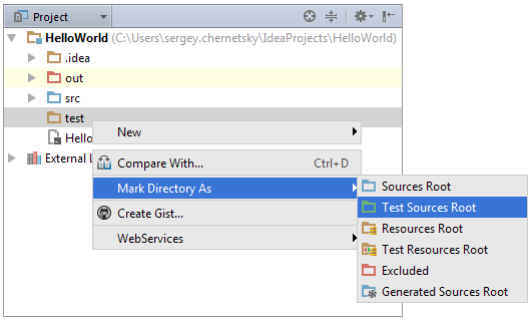
1. In the right-hand pane, under Source Folders, Resource Folders, Test Source Folders or Test Resource Folders, click to the right of the necessary folder (folder path).



2. In the dialog that opens, select For generated sources or For generated resources and click OK.

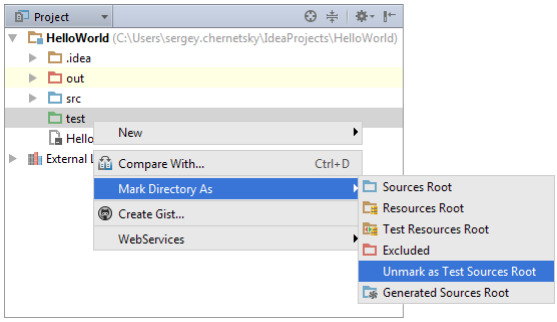
Using the Project tool window

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the folder of interest, point to Mark Directory As and select the desired category.



In a similar way you can change the folder category: just select a different category from the context menu.

To cancel the folder association with its current category (i.e. to make a folder an "ordinary" folder), select Unmark as <current_category> Root (e.g. Unmark as Test Sources Root).

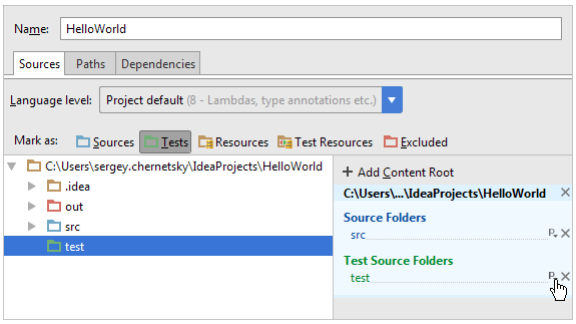


Assigning a package prefix to a folder with Java sources

In Java modules, you can assign a package prefix to a folder containing source files (e.g. `com.mycompany.myapp`). This eliminates the necessity to create the corresponding folder structure in that folder (e.g. `com/mycompany/myapp`). For more information, see Package prefix for Java source roots.

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, select Modules.
3. In the pane to the right, select the necessary module.

4. In the right-hand part of the dialog, select the Sources tab.
5. In the right-hand pane, under Source Folders or Test Source Folders, click to the right of the necessary folder (folder path).

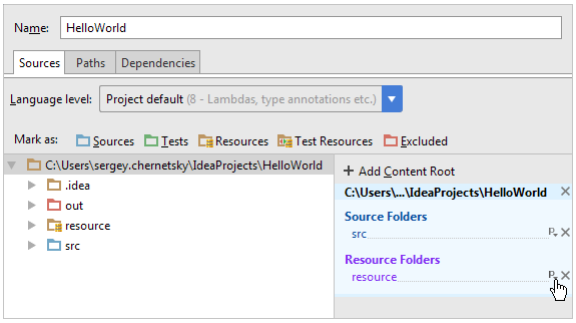


6. In the dialog that opens, specify the package prefix and click OK.
7. Click OK in the Project Structure dialog.

Specifying the output path for resources

During the build process, by default, the resources are copied into the root of the compilation output folder. If necessary, you can specify a different folder within that output folder.

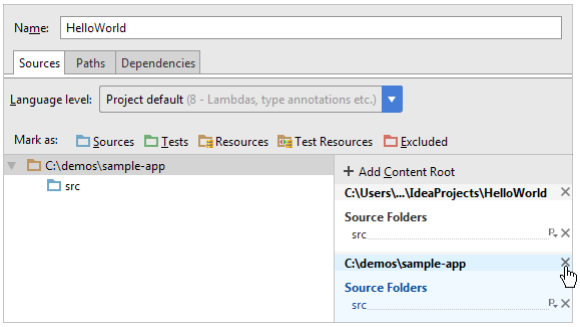
1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, select Modules.
3. In the pane to the right, select the necessary module.
4. In the right-hand part of the dialog, select the Sources tab.
5. In the right-hand pane, under Resource Folders or Test Resource Folders, click to the right of the necessary folder (folder path).



6. In the dialog that opens, specify the path relative to the output folder root, and click OK.
7. Click OK in the Project Structure dialog.

Removing a content root

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, select Modules.
3. In the pane to the right, select the necessary module.
4. In the right-hand part of the dialog, select the Sources tab.
5. To the right of the content root that you want to remove, click .



6. Confirm you intention to remove the content root. (The corresponding folder won't be physically deleted.)
7. Click OK in the Project Structure dialog.

Excluding Files from Project

On this page:

– Basics
– Marking a file as plain text
– Marking a file with its regular type
– Excluding a .phar archive from a project

Basics

Sometimes you might need to exclude a single file from your project, so that it will be ignored by inspections, code completion, etc. This is done using the Mark as plain text action.

When a file is marked as plain text, IntelliJ IDEA does not use it anymore for code completion and navigation. Such file is shown as plain text in the editor, and is denoted with a special icon in the Project tool window .

Note This action is not applicable to Java files and binaries.

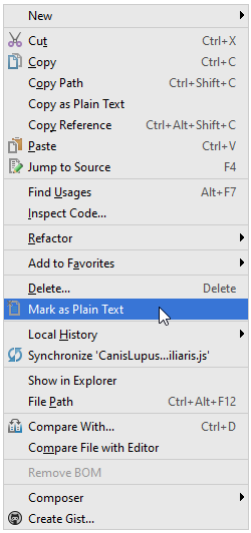
The reverse action is also available: you can return a file to its original type, using the Mark as <file type> action.

In the context of a PHP project, you can also exclude entire PHP archives (`.phar` files) to prevent multiple declaration conflicts between classes in a `.phar` archive and the corresponding classes in the installed vendors directory.

Marking a file as plain text

To mark a file as plain text, follow these steps

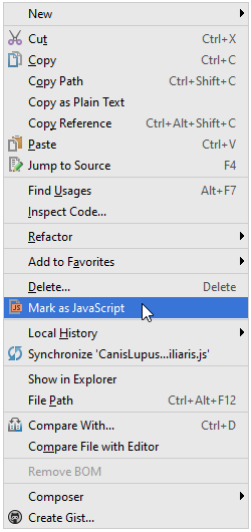
1. In the Project tool window , select the desired file.
2. On the context menu of the selection, choose Mark as plain text:



Marking a file with its regular type

To mark a file with its regular type, follow these steps

1. In the Project tool window , select the desired file, marked with icon.
2. On the context menu of the selection, choose Mark as <file type>:



Excluding a .phar archive from a project

To exclude a .phar archive from a project

- In the Project tool window, select the required `.phar` archive and choose Exclude phar from project on the context menu of the selection.

Configuring Module Dependencies and Libraries

On this page:

- Creating a module library and adding it to module dependencies
- Adding existing libraries to module dependencies
- Creating a module, project or global library and adding it to module dependencies
- Creating a library in the Project tool window
- Adding dependencies on modules
- Changing the order of dependencies
- Sorting the list of dependencies
- Configuring the library contents
- Moving a module library to the project or global level
- Downloading missing library sources from the Maven repository

See also, Dependencies , Library and Configuring Project and Global Libraries .

Creating a module library and adding it to module dependencies

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Modules.
3. In the pane to the right, select the module of interest.
4. In the right-hand part of the dialog, on the Module page, select the Dependencies tab.
5. On the Dependencies tab, click + and select Jars or directories.
6. In the dialog that opens , select the necessary files and folders. These may be individual `.class` and `.java` files, directories and archives (`.jar` and `.zip`) containing such files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`). Click OK.
7. If necessary, select the Export option and change the dependency scope .
8. Click OK in the Project Structure dialog.

See also, Adding existing libraries to module dependencies and Adding a project or global library to module dependencies.

Adding existing libraries to module dependencies

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Modules.
3. In the pane to the right, select the module of interest.
4. In the right-hand part of the dialog, on the Module page, select the Dependencies tab.
5. On the Dependencies tab, click + and select Library.
6. In the Choose Libraries dialog, select one or more libraries and click Add Selected.
   The libraries you can choose from include:
   - Global libraries
   - Project libraries existing in the current project
   - Application server libraries for the servers registered in IntelliJ IDEA
7. If necessary, select the Export option and change the dependency scope .
8. Click OK in the Project Structure dialog.

See also, Adding a project or global library to module dependencies.

Creating a module, project or global library and adding it to module dependencies

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Modules.
3. In the pane to the right, select the module of interest.
4. In the right-hand part of the dialog, on the Module page, select the Dependencies tab.
5. On the Dependencies tab, click + and select Library.
6. In the Choose Libraries dialog, click New Library.
7. If suggested, select:
   - Java to create a Java library. Select the files and folders to be included in the library in the dialog that opens .
   - From Maven to download a Java library from a public Maven repository . Specify the artifact to be downloaded and the associated settings in the Download Library from Maven Repository dialog.
8. In the Configure Library dialog :
   1. Specify the library name.

2. Select the library level (global, project or module).
3. Configure the library contents using +, [redacted] and –.
4. Click OK.

9. In the Choose Libraries dialog, click Add Selected.
10. If necessary, select the Export option and change the dependency scope .
11. Click OK in the Project Structure dialog.

See also, Creating a project or a global library and Adding a project or global library to module dependencies.

Creating a library in the Project tool window

Sometimes, you don't need to open the Project Structure dialog to create a library and add it to dependencies of a module. This is the case when there are `.jar` files within your project content roots .

You can select such `.jar` files in the Project tool window and use these files to create a library. When doing so, you will be able to select the library level and the module to whose dependencies the new library should be added.

In a similar way, you can use a directory containing `.jar` files to create a library.

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Select the `.jar` file or files to be included in the library, or a directory that contains the `.jar` files of interest.
3. Select Add as Library from the context menu.
4. In the Create Library dialog, specify the library name, level and the module in which this library will be used.

Adding dependencies on modules

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Modules.
3. In the pane to the right, select the module of interest.
4. In the right-hand part of the dialog, on the Module page, select the Dependencies tab.
5. On the Dependencies tab, click + and select Module Dependency.
6. In the Choose Modules dialog, select one or more modules the current module should depend on, and click OK. (Your selection is limited to the modules of the current project. That is, you cannot select the modules from a different project.)
7. If necessary, select the Export option and change the dependency scope .
8. Click OK in the Project Structure dialog.

Changing the order of dependencies

In certain cases, the order of module dependencies may be important, see Order of dependencies.

To reorder the items, use ↑ and ↓. Clicking these icons moves the selected item one line up or down in the list.

Sorting the list of dependencies

You can sort the dependencies by their names and scopes by clicking the cells in the header row.

If you click a cell once, the list is sorted by the corresponding column in the ascending order. The sorting marker appears in the cell: ▲. When you click the cell for the second time, the information is sorted in the descending order. To show this, the sorting marker changes its appearance: ▼. Finally, when you click the cell for the third time, the initial unsorted state is resorted.

Note that the sorting operations don't change the actual order of dependencies.

When the list is sorted, the icons for changing the order of dependencies are inactive.

Configuring the library contents

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Modules.
3. In the pane to the right, select the module of interest.
4. In the right-hand part of the dialog, on the Module page, select the Dependencies tab.
5. On the Dependencies tab, select the library of interest, and click ✎.
6. In the Configure Library dialog that opens, use +, [redacted] and – to manage the library contents.
7. Click OK in the Project Structure dialog.

Moving a module library to the project or global level

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Modules.
3. In the pane to the right, select the module of interest.
4. In the right-hand part of the dialog, on the Module page, select the Dependencies tab.
5. On the Dependencies tab, right-click the library of interest, and select Move to Project Libraries or Move to Global Libraries.
6. In the Move Library dialog, if necessary, edit the library name. If you want to change the location of the library files, select the Move library files to check box and specify the corresponding location.

Note that the corresponding library is not removed from the list of the module dependencies.

Downloading missing library sources from the Maven repository

For the library files originating form the Maven public repository, when missing sources are detected, IntelliJ IDEA suggests downloading and including those sources in the library.

To access these features, IntelliJ IDEA provides the corresponding hyperlinks in the editor.

Configuring Module Compiler Output

Use this setting to define the folders where the IntelliJ IDEA will place the compiled classes and test classes, regardless of the number of source and test roots in your module.

To configure module compiler output:

1. Open module settings dialog .
2. Select the desired module, and click the Paths tab .
3. In the Compiler output section, specify which output path will the compiler use. You can opt to direct the compiler output to the path, configured on the project level. If you want to use a module-specific output path, click the radio-button Use module compiler output path.
4. In the Output path field, specify the Production output path.
   Accept default, or click the ... button and locate the desired folder in the Select Path dialog.
5. In the Test output path field, specify the Test output path.
   Accept default, or click the ... button and locate the desired folder in the Select Path dialog.

Configuring Project Compiler Output

Project compiler output path defines the folder that stores all project compilation results. This folder contains Production and Test subfolders for production code and test sources respectively. Compiler output paths for each module within the project by default reside under the project output path. You can configure module-specific compiler output path for each module separately.

To configure compiler output on the project level

1. In the Project Structure dialog , select the Project node.
2. In the section Project compiler output specify the desired path.
   You can type the path manually in the text field, or navigate to the desired location using the Select Path dialog.

> Tip To open the Select Path dialog, press ... or `Shift+Enter` .

Configuring Global, Project and Module SDKs

To find out how various Software Development Kits (SDKs) are used in IntelliJ IDEA, see SDK .

On this page:

– Configuring SDKs at the global (IDE) level
– Configuring a project SDK
– Configuring a module SDK
– Making Apple JDK sources accessible in IntelliJ IDEA on OS X

To configure SDKs at the global (IDE) level

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, under Platform Settings, click SDKs.
3. To add a new SDK, click + and select the desired SDK type .
4. In the dialog that opens , select the SDK home directory and click OK.
   As a result, a new SDK is added to IntelliJ IDEA, and its settings are shown on the SDK page in the right-hand part of the dialog.
5. Optionally, edit the SDK name and contents .
6. If necessary, add more SDKs as described above.
7. Click OK in the Project Structure dialog.

To configure a project SDK

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, under Project Settings, click Project.
3. On the page that opens in the right-hand part of the dialog, select the necessary SDK from the Project SDK list.
4. If the desired SDK is not present in the list, click New and select the necessary SDK type .
5. In the dialog that opens , select the SDK home directory and click OK.
   As a result, a new SDK is added to IntelliJ IDEA and selected as the project SDK.
6. To view or edit the SDK name and contents , click Edit. (The SDK page will open.)
7. Click OK in the Project Structure dialog.

To configure a module SDK

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane, under Project Settings, click Modules.
3. In the area under + − | | | , select the module of interest.
4. In the right-hand part of the dialog, on the Module page, select the Dependencies tab .
5. Select the SDK from the Module SDK list. (To select the project SDK, select Project SDK. Note that if you change the project SDK later, the module SDK will change accordingly.)
6. If the desired SDK is not present in the list, click New and select the necessary SDK type .
7. In the dialog that opens , select the SDK home directory and click OK.
   As a result, a new SDK is added to IntelliJ IDEA and associated with the module.
8. To view or edit the SDK name and contents , click Edit. (The SDK page will open.)
9. Click OK in the Project Structure dialog.

To make Apple JDK sources accessible in IntelliJ IDEA on OS X

The Apple JDK included in OS X installation doesn't contain the JDK sources. So to make those sources accessible in IntelliJ IDEA, you should download and install the Java for OS X Developer Package and then add the JDK sources contained in this package (`src.jar`) to the definition of the corresponding JSDK in IntelliJ IDEA.

1. In a Web browser, go to the Downloads for Apple Developers page .
2. Sign in to the downloads page using your Apple developer account (Apple ID and password).
3. Download and install the Java for OS X Update Developer Package suitable for the OS X version that you are using.
4. In IntelliJ IDEA, open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
5. In the left-hand pane, under Platform Settings, select SDKs.
6. In the second from the left pane, select the Apple JDK of interest. (If the corresponding JDK is not present in the list, add its definition, see Configuring SDKs at the global (IDE) level.)

7. On the SDK page shown in the right-hand part of the dialog, select the Sourcepath tab and click + (`Alt+Insert`).

8. In the dialog that opens , select the file `src.jar` and click OK.
   (Usually, this file is located in

   `/Library/Java/JavaVirtualMachines/<java.version>/Contents/Home`

   where `<java.version>` is your JDK version, for example, `1.6.0_31-b04-415.jdk` or something similar.)

9. Click OK in the Project Structure dialog.

Configuring Project and Global Libraries

On this page:

- Creating a project or a global library
- Adding items to a project or global library
- Removing items from a project or global library
- Specifying online documentation URL for a project or global library
- Making items excluded in a project or global library
- Cancelling the excluded status of items in a project or global library
- Creating a copy of a project or global library
- Creating a copy of a global library at the project level
- Moving a project library to a global level
- Adding a project or global library to module dependencies
- Finding usages of a project or global library in a project
- Deleting project and global libraries

See also, Library , Configuring Module Dependencies and Libraries and Configuring JavaScript Libraries .

Creating a project or a global library

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. Depending on the library level , in the left-hand pane of the dialog, select:
   - Libraries if you are going to create a project library.
   - Global Libraries if you are going to create a global library.
3. Above the pane to the right, click + and select one of the following options:
   - Java. Select this option if you are creating a Java library.
   - JavaScript. Select this option if you are creating a JavaScript library. Note that this option is not available at the project level.
   - From Maven. Select this option if you are going to download a Java library from a public Maven repository .
   - ActionScript/Flex. Select this option if you are creating an ActionScript/Flex library. This option is always present at the global level. A project ActionScript/Flex library can be created only if there is a Flash module in your project.
4. If you are creating a Java, JavaScript or ActionScript/Flex library, the Select Library Files dialog opens. Select the files and folders that you want to be included in the library.
   If appropriate, the Detected Roots dialog is displayed showing potential library items. Use the check boxes to deselect the unnecessary items.
5. If you have selected the From Maven option, the Download Library from Maven Repository is shown. Specify the artifact to be downloaded and the destination directory. (To search for the artifact, you can use a class name, a keyword or a search pattern, or Maven coordinates .) Optionally, select Sources and/or JavaDocs for the source code and documentation to be included.
6. If you are creating a Java library, the Choose Modules dialog is shown. Select the modules in which the new library will be used.
7. If you are creating an ActionScript/Flex library, the Choose Flash Build Configurations dialog is shown. Select the build configurations in which the new library will be used.
8. If necessary, edit the library name and contents.
9. For a JavaScript library, specify its usage scope .
10. Click OK in the Project Structure dialog.

Adding items to a project or global library

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, depending on the library level , select Libraries or Global Libraries.
3. In the pane to the right, select the library of interest.
4. In the right-hand part of the dialog, on the Project Library or the Global Library page, click +. (For a JavaScript library, click + and select Attach Files or Directories.)
5. In the dialog that opens, select the files and folders that you want to be included in the library, and click OK.
6. Click OK in the Project Structure dialog.

Removing items from a project or global library

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, depending on the library level , select Libraries or Global Libraries.
3. In the pane to the right, select the library of interest.
4. In the right-hand part of the dialog, on the Project Library or the Global Library page, select the library items to be removed, and click −.
5. Click OK in the Project Structure dialog.

Specifying online documentation URL for a project or global library

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, depending on the library level , select Libraries or Global Libraries.
3. In the pane to the right, select the library of interest.
4. In the right-hand part of the dialog, on the Project Library or the Global Library page, click . (For a JavaScript library, click and select Specify Documentation URL instead.)
5. In the dialog that opens, specify the documentation URL and click OK.
6. Click OK in the Project Structure dialog.

See also, Viewing Reference Information .

Making items excluded in a project or global library

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, depending on the library level , select Libraries or Global Libraries.
3. In the pane to the right, select the library of interest.
4. In the right-hand part of the dialog, on the Project Library or the Global Library page, click .
5. In the dialog that opens, select the items that you want to make excluded, and click OK. (You can exclude folders, archives (e.g. JARs) and folders within the archives.)
   As a result, the items with the excluded status appear, the ones shown as .
6. Click OK in the Project Structure dialog.

See also, Excluded library items.

Cancelling the excluded status of items in a project or global library

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, depending on the library level , select Libraries or Global Libraries.
3. In the pane to the right, select the library of interest.
4. In the right-hand part of the dialog, on the Project Library or the Global Library page, select the items whose excluded status you want to cancel. (The items with the excluded status are shown as .)
5. Click .
6. Click OK in the Project Structure dialog.

See also, Excluded library items.

Creating a copy of a project or global library

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, depending on the library level , select Libraries or Global Libraries.
3. In the pane to the right, select the library of interest.
4. Click .
5. In the Copy Library dialog, specify the library name and click OK.
6. Click OK in the Project Structure dialog.

Creating a copy of a global library at the project level

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Global Libraries.
3. In the pane to the right, right-click the library of interest and select Copy to Project Libraries.
4. In the Copy Library dialog, specify the name of the library. If you want a copy of the library files to be created, select the Copy library files to check box and specify the corresponding location.
5. Click OK in the Project Structure dialog.

Moving a project library to a global level

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Libraries.

3. In the pane to the right, right-click the library of interest and select Move to Global Libraries.
4. In the Move Library dialog, if necessary, change the library name. If you want to change the location of the library files, select the Move library files to check box and specify the corresponding location.
5. Click OK in the Project Structure dialog.

Adding a project or global library to module dependencies

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, depending on the library level , select Libraries or Global Libraries.
3. In the pane to the right, right-click the library of interest and select Add to Modules.
4. In the Choose Modules dialog, select the modules in which the library will be used, and click OK.
5. Click OK in the Project Structure dialog.

See also, Dependencies and Adding existing libraries to module dependencies.

Finding usages of a project or global library in a project

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, depending on the library level , select Libraries or Global Libraries.
3. In the pane to the right, select the library of interest.
4. Click .

Deleting project and global libraries

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Libraries or Global Libraries.
3. In the pane to the right, select the libraries to be deleted.
4. Click .
5. Click OK in the Project Structure dialog.

Managing Facets

Facets let you extend the capabilities of your modules by adding support for corresponding technologies, frameworks and languages.

You can create the necessary facets when creating a project or a module. Facets can also be added to existing modules.

IntelliJ IDEA is capable of auto-detecting facets. As soon as it finds a file or a directory typical of a certain framework or technology, it suggests that you create the corresponding facet.

In this part:

– Adding and Removing Facets
– Changing Facet Settings
– Adding Auto-Detected Facets
– Configuring Auto-Detection of Facets
– Changing Default Facet Settings
– Changing the Settings for All Facets of the Same Type

Adding and Removing Facets

On this page:

– Introduction

– Adding a facet to a module

– Removing a facet from a module

Introduction

You can add facets to modules as well as remove them.

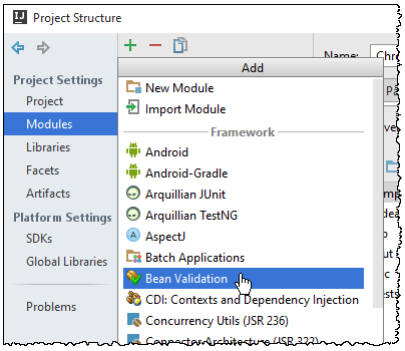
When adding facets to a module, take into consideration the facet dependencies , and also the possibility to add multiple instances of the same facet.

–

Adding a facet to a module

To add a facet to a module

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. Under Project Settings, select Modules.
3. Select the module you want to add a facet to, click +, and select the facet you want to add.



Note that certain facets have other facets as their parents. To add such facets, you should select a parent facet (rather than the module) prior to clicking +.

For example, a parent facet for a JSF or a Struts facet is a Web facet. So a JSF or Struts facet can be added only if a Web facet is currently selected. For more information, see Available Facets and Their Dependencies .

4. Configure the facet settings , and click OK in the Project Structure dialog.

Removing a facet from a module

To remove a facet from a module

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. Under Project Settings, select Modules.
3. Expand the module node and select the facet to be removed.
4. Click –.
5. Click OK in the Project Structure dialog.

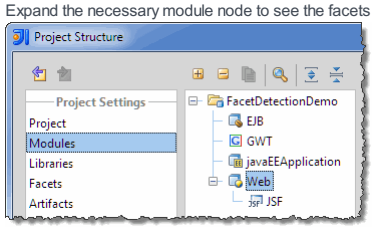
Note that no files are physically deleted as a result.

Changing Facet Settings

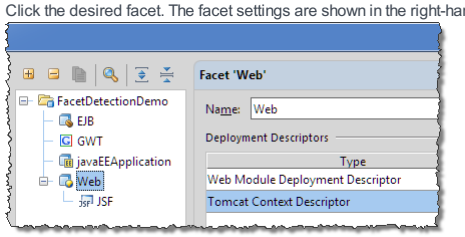
Facet settings vary depending on the facet types. This section provides only general instructions for changing these settings. Facet-specific options are discussed elsewhere .

To change facet settings

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. In the left-hand pane, click Modules.
3. Expand the necessary module node to see the facets associated with this module.



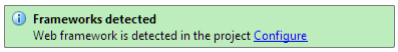
4. Click the desired facet. The facet settings are shown in the right-hand part of the dialog.



5. Modify the facet settings as required and click OK.

Adding Auto-Detected Facets

IntelliJ IDEA is capable of auto-detecting facets. As soon as it finds a file or a directory typical of a certain framework or technology, you get a notification looking similar to this:



By clicking Configure, you can start creating the corresponding facet or facets.

This task can also be initiated from within the Event Log tool window , which will contain a similar notification message.

To create detected facets

1. Click Configure in the notification box.
   Alternatively, open the Event Log tool window, find the corresponding notification message there, and click Configure within the message.

   To open the Event Log tool window, click on the status bar or Event Log on the tool window bar. Both these controls are located in the lower-right corner of the main window.

2. In the Setup Frameworks dialog that opens, select the facets to be created and the files to be associated with these facets.
   (The facets are at the top hierarchical level in this dialog; the corresponding files are shown as facet elements, at the second level.)

   Click OK.

Configuring Auto-Detection of Facets

On this page:

- Introduction
- Excluding facets from auto-detection
- Restoring auto-detection of facets

Introduction

Facet auto-detection is enabled for all facet types unless you exclude unnecessary types from auto-detection. This exclusion can be project-wide or limited to a certain directory.

If necessary, you can restore auto-detection for the facet types that you have excluded from auto-detection.

See also, Adding Auto-Detected Facets .

Excluding facets from auto-detection

To exclude a facet type from auto-detection

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. In the left-hand pane, click Facets.
3. In the area under + −, select Detection.
4. In the right-hand part of the dialog, click +.
5. Select the facet type, and then select where auto-detection should be disabled:
   - In the whole project.
   - In directory. Select the directory where auto-detection should be disabled in the dialog that opens .

   Note that you can disable auto-detection for all the available facet types at once by selecting the All Frameworks option.
6. Click OK in the Project Structure dialog.

Restoring auto-detection of facets

To restore auto-detection of facets

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. In the left-hand pane, click Facets.
3. In the area under + −, select Detection.
4. In the right-hand part of the dialog, select the facet types for which you want to restore auto-detection, and click −.
5. Click OK in the Project Structure dialog.

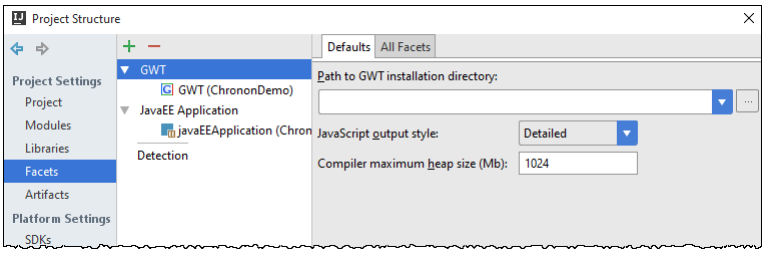
Changing Default Facet Settings

For certain facet types (e.g. GWT), you can specify default settings. These settings are applied when a new facet of the corresponding type is created.

Note that there are no default settings for most of the facet types.

To change default facet settings

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. In the left-hand pane, click Facets.
3. In the central pane, select the required facet type.
   Note that the facet types are shown for existing facets only, and appear at the top hierarchical level. The existing facets are shown within the corresponding type, at the second level.
4. The default settings (if any) are shown in the right-hand part of the dialog. If there is the Defaults tab there, this is the one that you want.



   Configure the default settings as necessary.

5. Click Apply or OK in the Project Structure dialog.

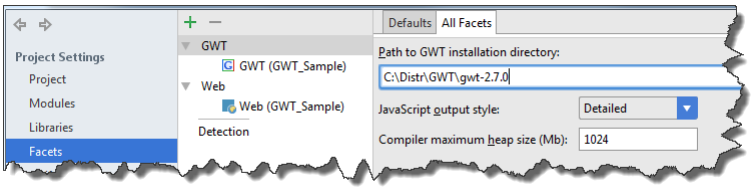
Changing the Settings for All Facets of the Same Type

For certain facet types (e.g. GWT), you can change certain settings for all the existing facets at once.

Note that for most of the facet types this feature is not available.

To change the settings for all facets of the same type

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. In the left-hand pane, click Facets.
3. In the central pane, select the required facet type.
   Note that the facet types are shown for existing facets only, and appear at the top hierarchical level. The existing facets are shown within the corresponding type, at the second level.
4. In the right-hand part of the dialog, select the All Facets tab (if present).



   Configure the settings as necessary.

5. Click Apply or OK in the Project Structure dialog.

File and Code Templates

In this part:

- File and Code Templates
  - Overview
  - Per-project vs default scheme
  - Predefined, internal, and custom templates
  - When are file and code templates used?
  - Are the choices of templates context-sensitive?
- File Template Variables
- #Parse Directive
- Creating and Editing File Templates

Overview

File templates are specifications of the contents to be generated when creating a new file. They let you create the source files that already contain some initial code.

You can view, edit and create the templates on the File and Code Templates page of the Settings/ Preferences dialog.

File and code templates are written in the Velocity Template Language (VTL).

So they may include:

- Fixed text (markup, code, comments, etc.). In a file based on a template, the fixed text is used literally, as-is.
- File template variables . When creating a file, the variables are replaced with their values.
- #parse directives to include other templates defined in the Includes tab on the File and Code Templates page of the Settings/Preferences dialog box.
- Other VTL constructs.

Here is a typical template example. (This template is used for creating a Java class.)

```
#if (${PACKAGE_NAME} != "")package ${PACKAGE_NAME};#end
                    #parse("File Header.java")
                    public class ${NAME} {
                    }
```

In this template:

- `${PACKAGE_NAME}` and `${NAME}` are template variables.
- The `#parse` directive is used to include the other template `File Header.java` .
- The first line contains an `#if` VTL directive.

Applying this template leads to generating a file whose contents look similar to this:

```
package demo;

                    /**
                     * Created by IntelliJ IDEA.
                     * User: John.Smith
                     * Date: 6/1/11
                     * Time: 12:54 PM
                     * To change this template use File | Settings | File and Code Templates.
                     */
                    public class Demo {
                    }
```

Per-project vs default scheme

IntelliJ IDEA suggests using file and code templates on the project or default (global) level.

If you need a sharable set of file and code templates, then these templates should be per-project; otherwise the templates are global and pertain to the entire workspace.

The file and code templates are stored in the following locations:

- The default (global) templates are stored in the IntelliJ IDEA home directory, in the folder `config | fileTemplates` .
- The per-project file and code templates are stored in the `.idea | fileTemplates` folder. These templates can be shared among the team members.

Refer to the section Project and IDE Settings to learn where the settings are stored, and to the File and Code Templates dialog for the description of the Schema field.

Predefined, internal, and custom templates

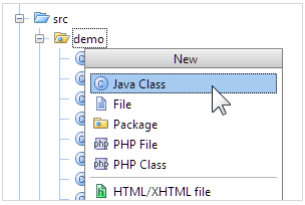
IntelliJ IDEA comes with a set of predefined file and code templates. You can use these templates as-is or modify them as necessary. You can as well create your own templates (custom templates).

Internal file and code templates are a subset of the predefined templates. These templates differ from all the other templates in that they cannot be deleted.

On the File and Code Templates page of the Settings/Preferences dialog, the names of internal templates are shown in bold. The names of the custom templates and the predefined templates that you have modified are shown in blue.

When are file and code templates used?

Whenever you create a new file, you can choose to create an empty file (e.g. File | New | File) or use a file template. In the latter case, the initial contents of the new file will be generated according to the template you have selected. (Basically, all the options in the New menu except File , Package and Directory correspond to using a template.)



Are the choices of templates context-sensitive?

Generally, the set of the templates you can choose from when creating a new file depends on your module nature and configuration as well as the properties of your current location in the Project tool window. For example, you are not offered to use a template for an ActionScript class or interface, or an MXML component if your module is not a Flash module. Similarly, you cannot choose to create a Java class, interface, etc. outside of Java source or test directories (roots).

However, there are many templates that are available in any context.

File Template Variables

On this page:

– Basics
– Predefined template variables
– Custom template variables

Basics

A file template can contain template variables. When a template is applied, the variables are replaced with their values.

A file template variable is a string that starts with a dollar sign which is followed by the variable name. The variable name may be enclosed in curly braces. For example: `$MyVariable` or `${MyVariable}`.

Predefined template variables

IntelliJ IDEA comes with a set of predefined template variables.

The available predefined file template variables are:

– `${PACKAGE_NAME}` - the name of the target package where the new class or interface will be created.
– `${PROJECT_NAME}` - the name of the current project.
– `${FILE_NAME}` - the name of the PHP file that will be created.
– `${NAME}` - the name of the new file which you specify in the New File dialog box during the file creation.
– `${USER}` - the login name of the current user.
– `${DATE}` - the current system date.
– `${TIME}` - the current system time.
– `${YEAR}` - the current year.
– `${MONTH}` - the current month.
– `${DAY}` - the current day of the month.
– `${HOUR}` - the current hour.
– `${MINUTE}` - the current minute.
– `${PRODUCT_NAME}` - the name of the IDE in which the file will be created.
– `${MONTH_NAME_SHORT}` - the first 3 letters of the month name. Example: Jan, Feb, etc.
– `${MONTH_NAME_FULL}` - full name of a month. Example: January, February, etc.

IntelliJ IDEA provides a set of additional variables for PHP include templates , that is, templates of reusable fragments that can be included in other PHP file template. The built-in PHP include templates are intended for generating file headers and PHPDoc documentation comments . The following variables are available in PHP include templates:

– `${NAME}` - the name of the class, field, or function (method) for which the PHPDoc comment will be generated.
– `${NAMESPACE}` - the fully qualified name (without a leading slash) of the class or field namespace.
– `${CLASS_NAME}` - the name of the class where the field to generate the PHPDoc comment for is defined.
– `${STATIC}` - gets the value `static` if the function (method) or field to generate the comment for is `static`. Otherwise evaluates to an `empty string`.
– `${TYPE_HINT}` - a prompt for the `return` value of the function (method) to generate the comment for. If the return type cannot be detected through the static analysis of the function (method), evaluates to `void`.
– `${PARAM_DOC}` - - a documentation comment for parameters. Evaluates to a set of lines `@param type name`. If the function to generate comments for does not contain any parameters, the variable evaluates to empty content.
– `${THROWS_DOC}` - a documentation comment for exceptions. Evaluates to a set of lines `@throws type`. If the function to generate comments for does not throw any exceptions, the variable evaluates to empty content.
– `${DS}` - a dollar character (`$`). The variable evaluates to a plain dollar character (`$`) and is used when you need to escape this symbol so it is not treated as a prefix of a variable.
– `${CARET}` - indicated the position of the caret after generating and adding the comment.
This `${CARET}` variable is applied only when a PHPDoc comment is generated and inserted during file creation. When a PHPDoc comment is created through Code | Generate | PHPDoc block, multiple selection of functions or methods is available so documentation comments can be created to several classes, functions, methods, or fields. As a result, IntelliJ IDEA cannot "choose" the block to apply the `${CARET}` variable in, therefore in this case the `${CARET}` variable is ignored.

Custom template variables

In addition to the predefined template variables, it is possible to specify custom variables. If necessary, you can define the values of custom variables right in a template using the `#set` VTL directive.

For example, if you want to use your full name instead of your login name defined through the pre-defined variable `${USER}`, write the following construct:

```
#set( $MyName = "John Smith" )
```

If the value of a variable is not defined in the template, IntelliJ IDEA will ask you to specify it when the template is applied.

You can prevent treating dollar characters (`$`) in template variables as prefixes. If you need a dollar character (`$` inserted as is, use the `${DS}` file template variable instead. When the template is applied, this variable evaluates to a plain dollar character (`$`).

#parse Directive

Using the `#parse` directive, you can include other templates in file templates . This is useful for inserting reusable contents (e.g. standard headers, copyright statements, etc.) into multiple file templates.

The syntax for the `#parse` directive is:

```
#parse("<template_name.extension>")
```

For example: `#parse("File Header.java")` .

The templates that can be referenced like this in other templates, are shown on the Includes tab of the File and Code Templates settings page.

Creating and Editing File Templates

IntelliJ IDEA supports several ways of creating file templates :

- Creating a file template from scratch
- Creating a file template from an existing one
- Creating a file template from a file
- Creating and referencing include templates

Creating a file template from scratch

1. Open Settings/Preferences dialog and under the Editor node, select File and Code Templates page .
2. Switch to the Files tab.
3. Click + on the toolbar and specify the template name, file extension, and the body of the template, which can contain:
   1. Plain text.
   2. `#parse` directives to work with includes .
   3. Predefined variables to be expanded into corresponding values in the format `${<variable_name>}` .
      The available predefined file template variables are:
      - `${PACKAGE_NAME}` - the name of the target package where the new class or interface will be created.
      - `${PROJECT_NAME}` - the name of the current project.
      - `${FILE_NAME}` - the name of the PHP file that will be created.
      - `${NAME}` - the name of the new file which you specify in the New File dialog box during the file creation.
      - `${USER}` - the login name of the current user.
      - `${DATE}` - the current system date.
      - `${TIME}` - the current system time.
      - `${YEAR}` - the current year.
      - `${MONTH}` - the current month.
      - `${DAY}` - the current day of the month.
      - `${HOUR}` - the current hour.
      - `${MINUTE}` - the current minute.
      - `${PRODUCT_NAME}` - the name of the IDE in which the file will be created.
      - `${MONTH_NAME_SHORT}` - the first 3 letters of the month name. Example: Jan, Feb, etc.
      - `${MONTH_NAME_FULL}` - full name of a month. Example: January, February, etc.

      IntelliJ IDEA provides a set of additional variables for PHP include templates , that is, templates of reusable fragments that can be included in other PHP file template. The built-in PHP include templates are intended for generating file headers and PHPDoc documentation comments . The following variables are available in PHP include templates:
      - `${NAME}` - the name of the class, field, or function (method) for which the PHPDoc comment will be generated.
      - `${NAMESPACE}` - the fully qualified name (without a leading slash) of the class or field namespace.
      - `${CLASS_NAME}` - the name of the class where the field to generate the PHPDoc comment for is defined.
      - `${STATIC}` - gets the value `static` if the function (method) or field to generate the comment for is `static` . Otherwise evaluates to an `empty string` .
      - `${TYPE_HINT}` - a prompt for the `return` value of the function (method) to generate the comment for. If the return type cannot be detected through the static analysis of the function (method), evaluates to `void` .
      - `${PARAM_DOC}` - - a documentation comment for parameters. Evaluates to a set of lines `@param type name` . If the function to generate comments for does not contain any parameters, the variable evaluates to empty content.
      - `${THROWS_DOC}` - a documentation comment for exceptions. Evaluates to a set of lines `@throws type` . If the function to generate comments for does not throw any exceptions, the variable evaluates to empty content.
      - `${DS}` - a dollar character (`$`). The variable evaluates to a plain dollar character (`$`) and is used when you need to escape this symbol so it is not treated as a prefix of a variable.
      - `${CARET}` - indicated the position of the caret after generating and adding the comment.
        This `${CARET}` variable is applied only when a PHPDoc comment is generated and inserted during file creation. When a PHPDoc comment is created through Code | Generate | PHPDoc block, multiple selection of functions or methods is available so documentation comments can be created to several classes, functions, methods, or fields. As a result, IntelliJ IDEA cannot "choose" the block to apply the `${CARET}` variable in, therefore in this case the `${CARET}` variable is ignored.
   4. Custom variables. Their names can be defined right in the template through the `#set` directive or will be defined during the file creation.
4. To have the dollar character (`$`) in a variable rendered "as is", use the `${DS}` variable instead. This variable evaluates to a plain dollar character (`$`).
5. Apply the changes and close the dialog box.

Creating a file template from an existing one

1. Open the File Templates settings page and switch to the Files tab.
2. Click on the toolbar and change the template name, extension, and source code as required.
3. Apply the changes and close the dialog box.

Creating a file template from a file

1. Open the desired file in the editor.
2. On the main menu, choose Tools | Save File as Template.

3. In the File and Code Templates dialog box that opens specify the new template name and edit the source code, if necessary.

4. Apply the changes and close the dialog box.

Creating and referencing include templates

Include templates are used to define reusable pieces of code to be inserted in file templates through `#parse` directives.

1. In the File and Code Templates settings page, switch to the Includes tab.

2. Click + on the toolbar and specify the template name, extension, and the source code. Do one of the following:
   - Use the predefined file template variables .
   - Create custom template variables and define their values right in the include template using the `#set` VTL directive.
     For example, if you want to your full name inserted in the file header instead of your login name defined through the `${USER}`, write the following construct:

     ```
     #set( $MyName = "John Smith" )
     ```

     If, when applying a template, the values of certain template variable are not known, IntelliJ IDEA will ask you to specify them.

   You can prevent treating dollar characters (`$`) in template variables as prefixes. If you need a dollar character (`$` inserted as is, use the `${DS}` file template variable instead. When the template is applied, this variable evaluates to a plain dollar character (`$`).

3. To use the include template, switch to the Templates tab, select the desired template and click Edit.

4. To include a template, insert the #parse directive in the source code.

Live Templates

In this part:

- Live Templates
  - Overview
  - Important notes
- Simple, Parameterized and Surround Live Templates
- Live Template Abbreviation
- Live Template Variables
- Groups of Live Templates
- Creating and Editing Live Templates
- Creating and Editing Template Variables
- Sharing Live Templates

Overview

Live templates let you insert frequently-used or custom code constructs into your source code file quickly, efficiently, and accurately.

Live templates are stored in the following location:

- Windows: `<your home directory>\.<product name><version number>\config\templates`
- Linux: `~/.<product name><version number>/config/templates`
- OS X: `~/Library/Preferences/<product name><version number>/templates`

Important notes

- To create or edit the Live templates, use the Live Templates page of the Settings/Preferences dialog.
- Having created a custom live template , you can export all the live templates together with the other settings, and import them. Refer to the section Exporting and Importing Settings for details.

Simple, Parameterized and Surround Live Templates

On this page:

– Simple live templates
– Parameterized live templates
– Surround live templates
– Examples

Simple live templates

Simple templates contain some fixed code that expands into plain text. When invoked and expanded in the editor, the code specified in the template is automatically inserted into your source code, replacing the abbreviation.

Parameterized live templates

Parameterized templates contain plain text and variables that enable user input.

Tip If you need a dollar sign (`$`) in the template text, escape it by duplicating this character (`$$`).

After a template is expanded, variables appear in the editor as input fields whose values can be either filled in by the user or calculated by IntelliJ IDEA automatically.

When a parameterized live template is invoked and expanded in the Editor, IntelliJ IDEA can suggest some predefined values at the input positions of the variables. For example, if a parameterized template contains code for an iteration, then, on expanding the template, IntelliJ IDEA will suggest:

– A name for the index variable (`i` , `j` , etc.).
– A list of all suitable variables in the current scope (e.g. Arrays) as an expression for the iterated container.
– A name for the assigned variable that holds the current container element.
– Type of the elements in the iterated container.

Surround live templates

Surround templates work only with blocks of selected text. Such templates place code before and after the selected block.

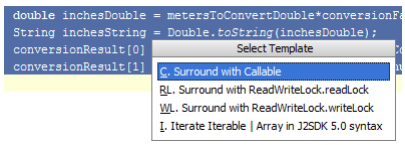
Examples

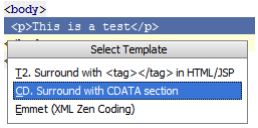
Insert parameterized live template (`Ctrl+J`):

```
for (int i = 0; i <_; i++) {

}
```

Refer also to the page Iterating Over an Array. Example of Applying Parameterized Live Templates .

Surround with live template (`Ctrl+Alt+J`):





Refer also to the page Wrapping a Tag. Example of Applying Surround Live Templates .

Live Template Abbreviation

Each live template is identified by a template abbreviation.

The template abbreviations work like shortcuts and are expanded into fragments of source code, depending on the surrounding context. So doing, IntelliJ IDEA can format the generated code fragments according to the code style settings.

An abbreviation may contain alphanumeric characters, dots, and hyphens, and must be unique within its group . However, you can use the same abbreviation for different templates provided that they are in different groups.

Whole code constructs can be created using the template abbreviations. To do that, type the template abbreviation, and press expansion key (for example, `Tab`). Refer to the section Creating Code Constructs by Live Templates for details.

Live Template Variables

On this page:

– What are template variables
– Declaring template variables
– Creating and editing template variables
– Predefined template variables
– Predefined functions to use in live template variables

What are template variables

Template variables in live templates enable user input. After a template is expanded, variables appear in the editor as input fields.

Declaring template variables

Variables within templates are declared in the following format:

```
$<variable_name>$
```

Creating and editing template variables

Variables are defined by expressions, and can accept some default values.

This expression may contain constructs of the following basic types:

– String constants in double quotes.
– The name of another variable defined in a live template.
– Predefined functions with possible arguments.

.

Template variables are editable in the Edit Template Variables Dialog , which contains a complete list of available functions. See the list of predefined functions below on this page.

Predefined template variables

IntelliJ IDEA supports two predefined live template variables: `$END$` and `$SELECTION$` .

You cannot edit the predefined live template variables `$END$` and `$SELECTION$` .

– `$END$` indicates the position of the cursor after the template is expanded. For example, the template `return $END$;` will be expanded into

```
return ;
```

with the cursor positioned right before the semicolon.

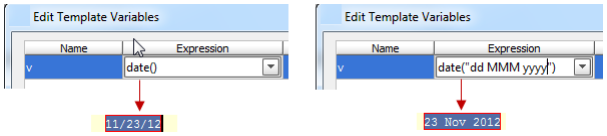
– `$SELECTION$` is used in surround templates and stands for the code fragment to be wrapped. After the template is expanded, the selected text is wrapped as specified in the template.
For example, if you select `EXAMPLE` in your code and invoke the `"$SELECTION$"` template via the assigned abbreviation or by pressing `Ctrl+Alt+T` and selecting the desired template from the list, IntelliJ IDEA will wrap the selection in double quotes as follows:

```
"EXAMPLE"
```

.

Predefined functions to use in live template variables

Item	Description
`annotated("annotation qname")`	Creates a symbol of type with an annotation that resides at the specified location. For an example, see Live Templates in the iterations group.
`arrayVariable()`	Suggests all array variables applicable in the current scope. For an example, see Live Templates in the iterations group.
`anonymousSuper()`	Suggests a supertype for a Kotlin object expression.
`camelCase(String)`	Returns the string passed as a parameter, converted to camel case. For example, `my-text-file` / `my text file` / `my_text_file` will be converted to `myTextFile` .
`capitalize(String)`	Capitalizes the first letter of the name passed as a parameter.
`capitalizeAndUnderscore(sCamelCaseName)`	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is `FooBar` , then the function returns `FOO_BAR` .
`castToLeftSideType()`	Casts the right-side expression to the left-side expression type. It is used in the iterations group to have a single template for generating both raw-type and Generics Collections.
`className(sClassName)`	Returns the name of the current class (the class where the template is expanded).
`classNameComplete()`	This expression substitutes for the class name completion at the variable position.

`clipboard()`	Returns the contents of the system clipboard.
`camelCase(String)`	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is `foo_bar`, then the function returns `fooBar`.
`complete()`	This expression substitutes for the code completion invocation at the variable position.
`completeSmart()`	This expression substitutes for the smart type completion invocation at the variable position.
`componentTypeOf (<array variable or array type>)`	Returns component type of an array. For example, see the Live Templates in the iterations group in the other group.
`currentPackage()`	Returns the current package name.
`date(sDate)`	Returns the current system date in the specified format. By default, the current date is returned in the default system format. However, if you specify date format in double quotes, the date will be presented in this format: 
`decapitalize(sName)`	Replaces the first letter of the name passed as a parameter with the corresponding lowercase letter.
`descendantClassEnum(<String>)`	Shows the children of the class entered as a string parameter.
`enum(sCompletionString1,sCompletionString2,...)`	List of comma-delimited strings suggested for completion at the template invocation.
`escapeString(sEscapeString)`	Escapes the specified string.
`expectedType()`	Returns the type which is expected as a result of the whole template. Makes sense if the template is expanded in the right part of an assignment, after return, etc.
`fileName(sFileName)`	Returns file name with extension.
`fileNameWithoutExtension()`	Returns file name without extension.
`firstWord(sFirstWord)`	Returns the first word of the string passed as a parameter.
`groovyScript("groovy code")`	Returns Groovy script with the specified code. You can use `groovyScript` macro with multiple arguments. The first argument is a script text that is executed or a path to the file that contains a script. The next arguments are bound to `_1, _2, _3, ..._n` variables that are available inside your script. Also, `_editor` variable is available inside the script. This variable is bound to the current editor.
`guessElementType (<container>)`	Makes a guess on the type of elements stored in a `java.util.Collection`. To make a guess, IntelliJ IDEA tries to find the places where the elements were added to or extracted from the container.
`iterableComponentType(<ArrayOrIterable>)`	Returns the type of an iterable component, such as an array or a collection.
`iterableVariable()`	Returns the name of a variable that can be iterated.
`lineNumber()`	Returns the current line number.
`lowercaseAndDash(String)`	Returns lower case separated by dashes, of the string passed as a parameter. For example, the string `MyExampleName` is converted to `my-example-name`.
`methodName()`	Returns the name of the embracing method (where the template is expanded).
`methodParameters()`	Returns the list of parameters of the embracing method (where the template is expanded).
`methodReturnType()`	Returns the type of the value returned by the current method (the method within which the template is expanded).
`qualifiedClassName()`	Returns the fully qualified name of the current class (the class where the template is expanded). Tip Clear the Shorten FQ names check box.
`rightSideType()`	Declares the left-side variable with a type of the right-side expression. It is used in the iterations group to have a single template for generating both raw-type and Generics Collections.
`snakeCase(sCamelCaseText)`	Returns snake_case string out of CamelCase string passed as a parameter.
`spaceSeparated(String)`	Returns string separated with spaces out of CamelCase string passed as a parameter. For example, if the string passed as a parameter is `fooBar`, then the function returns `foo bar`.
`subtypes(sType)`	Returns the subtypes of the type passed as a parameter.
`suggestIndexName()`	Suggests the name of an index variable. Returns `i` if there is no such variable in scope, otherwise returns `j` if there is no such variable in scope, etc.
`suggestVariableName()`	Suggests the name for a variable based on the variable type and its initializer expression, according to your code style settings that refer to the variable naming rules. For example, if it is a variable that holds an element within iteration, IntelliJ IDEA makes a guess on the most reasonable names, also taking into account the name of the container being iterated.
`suggestFirstVariableName(sFirstVariableName)`	Doesn't suggest `true, false, this, super`.
`time(sSystemTime)`	Returns the current system time.
`typeOfVariable(VAR)`	Returns the type of the variable passed as a parameter.
`underscoresToCamelCase(sCamelCaseText)`	Returns the string passed as a parameter with CamelHump letters substituting for underscores. For example, if the string passed as a parameter is `foo_bar`, then the function returns `fooBar`.
`underscoresToSpaces(sParameterWithSpaces)`	Returns the string passed as a parameter with spaces substituting for underscores.
`user()`	Returns the name of the current user.
`variableOfType(<type>)`	Suggests all variables that may be assigned to the type passed as a parameter, for example variableOfType("java.util.Vector"). If

	you pass an empty string ("") as a parameter, suggests all variables regardless of their types.
`JsArrayVariable`	Returns JavaScript array name.
`jsClassName()`	Returns the name of the current JavaScript class.
`jsComponentType`	Returns the JavaScript component type.
`jsMethodName()`	Returns the name of the current JavaScript method.
`jsQualifiedClassName`	Returns the complete name of the current JavaScript class.
`jsSuggestIndexName`	Returns a suggested name for an index.
`jsSuggestVariableName`	Returns a suggested name for a variable.

Groups of Live Templates

All templates are stored in groups for your convenience. Usually, templates are arranged into groups in accordance with the context they are sensitive to.

Each group is represented by an `xml` file with the name `<group>.xml`; for example, `css.xml`. These files are stored in the `config/templates` directory that resides in the IntelliJ IDEA's user settings directory.

IntelliJ IDEA comes with a set of pre-defined groups of live templates. The user group, represented by `user.xml` file, is intended for storing custom live templates.

Tip It is strongly recommended to avoid storing custom templates in the pre-defined IntelliJ IDEA groups. For this purpose, use the user group, or a new group.

Creating and Editing Live Templates

On this page:

- Introduction
- Modifying existing templates
- Creating a new live template from scratch
- Creating a new live template from a text fragment
- Searching through the list of live templates
- Restoring defaults

Introduction

IntelliJ IDEA comes with a set of the predefined Live Templates . You can use them as is, or modify them to suit your needs. If you want to create a new live template, you can do it from scratch, on the base of the copy of an existing template, or from a fragment of source code.

If a template has been changed, it is always possible to restore its default settings.

Modifying existing templates

To modify an existing template

1. In the Settings/Prefereces dialog, open the Live Templates page.
2. Expand the desired template group, and select the template you want to change.
3. In the Template Text area, change the template abbreviation as required.
4. In the Template Text field, edit the template body which may contain plain text and variables in the format `$<variable name>$` .
   When editing the live template variables, mind the following helpful hints:
   - If you need a dollar sign (`$`) in the template text, escape it by duplicating this character (`$$`).
   - To change the variables in a template, click the Edit Variables button and configure the variables as described in Creating and Editing Template Variables .

   The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from `$END$` or `$SELECTION$` .

   #### Side note about pe-defined template variables

   IntelliJ IDEA supports two predefined live template variables: `$END$` and `$SELECTION$` .

   You cannot edit the predefined live template variables `$END$` and `$SELECTION$` .
   - `$END$` indicates the position of the cursor after the template is expanded. For example, the template `return $END$;` will be expanded into

     ```
     return ;
     ```

     with the cursor positioned right before the semicolon.
   - `$SELECTION$` is used in surround templates and stands for the code fragment to be wrapped. After the template is expanded, the selected text is wrapped as specified in the template.
     For example, if you select `EXAMPLE` in your code and invoke the `"$SELECTION$"` template via the assigned abbreviation or by pressing `Ctrl+Alt+T` and selecting the desired template from the list, IntelliJ IDEA will wrap the selection in double quotes as follows:

     ```
     "EXAMPLE"
     ```

     .
5. In the Options section, specify how the template will be expanded and reformatted.
6. In the Available in section, specify the languages and places of code where the editor should be sensitive to the template abbreviation.
7. Click OK when ready.

Creating a new live template from scratch

To create a new template from scratch

1. In the Settings dialog, open the Live Templates page, and expand the template group where you want to create a new template.
2. Click +. A new template item is added to the group and the focus moves to the Template Text area.
3. Specify the new template abbreviation, type the template body, define the variables and the template group, configure the options, as described in the template modification procedure.
4. Click OK when ready.

Creating a new live template from a text fragment

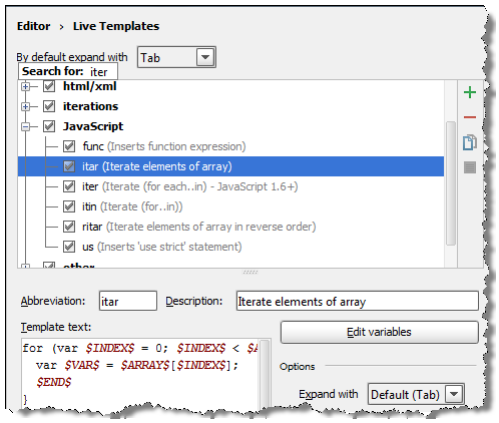
To create a live template from a text fragment

1. In the editor, select the text fragment to create a live template from.
2. On the main menu, choose Tools | Save as Live Template. The Live Templates page opens, with the Template Text area in focus.
3. In the Abbreviation field, type abbreviation to identify your new live template.
4. Specify the new template abbreviation, type the template body, define the variables and the template group, configure the options, as described in the template modification procedure.
5. Click OK when ready.

Searching through the list of live templates

To search through the list of live templates

– In the Live Templates page, start typing any string, which you expect to exist in the template abbreviation, body, or description.
IntelliJ IDEA shows all matching templates:



Restoring defaults

To restore default settings of a template

Note that a modified template is color-coded - it is shown blue.

1. In the Live Templates page, right-click a modified template to reveal its context menu.
2. Choose Restore defaults on the context menu of the modified template.

Creating and Editing Template Variables

On this page:

– Basics
– Configuring variables used in a template
– Predefined Functions to Use in Live Template Variables

Basics

After a template is expanded, its variables are presented in the editor as input fields. The values of these fields can be either filled in by the user or calculated by IntelliJ IDEA.

To have it done automatically, for each variable you need to specify the following:

– Expression to be calculated in association with the variable.
– Default value to be entered in the input field if the calculation fails.

The order in which IntelliJ IDEA will process input fields after the template expansion, is determined by the order of variables in the list.

Configuring variables used in a template

1. Open the template settings , and in the Template Text area click the Edit Variables button.
   The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from `$END$` or `$SELECTION$` .

   The Edit Template Variables dialog box opens, where you can define how the variables will be processed when the template is used.

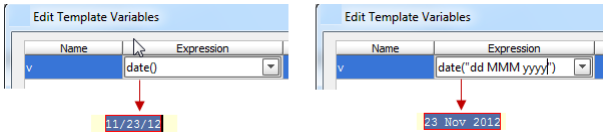
2. In the Name text box, specify the variable name to be used in the template body.

3. In the Expression drop-down list, specify the expression to be calculated by IntelliJ IDEA when the template is expanded. Do one of the following:
   – Type a string constant in double quotes.
   – Type a predefined function with possible arguments or select one from the drop-down list.
   An argument of a function can be either a line constant or another predefined function. See the list of predefined functions below on this page.

4. To enable IntelliJ IDEA to proceed with the next input field, if an input field associated with the current variable is already defined, select the Skip if defined check box.

5. To arrange variables in the order you want IntelliJ IDEA to switch between associated input fields, use the Move Up and Move Down buttons.

Predefined Functions to Use in Live Template Variables

Item	Description
`annotated("annotation qname")`	Creates a symbol of type with an annotation that resides at the specified location. For an example, see Live Templates in the iterations group.
`arrayVariable()`	Suggests all array variables applicable in the current scope. For an example, see Live Templates in the iterations group.
`anonymousSuper()`	Suggests a supertype for a Kotlin object expression.
`camelCase(String)`	Returns the string passed as a parameter, converted to camel case. For example, `my-text-file` / `my text file` / `my_text_file` will be converted to `myTextFile` .
`capitalize(String)`	Capitalizes the first letter of the name passed as a parameter.
`capitalizeAndUnderscore(sCamelCaseName)`	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is `FooBar` , then the function returns `FOO_BAR` .
`castToLeftSideType()`	Casts the right-side expression to the left-side expression type. It is used in the iterations group to have a single template for generating both raw-type and Generics Collections.
`className(sClassName)`	Returns the name of the current class (the class where the template is expanded).
`classNameComplete()`	This expression substitutes for the class name completion at the variable position.
`clipboard()`	Returns the contents of the system clipboard.
`camelCase(String)`	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is `foo_bar` , then the function returns `fooBar` .
`complete()`	This expression substitutes for the code completion invocation at the variable position.
`completeSmart()`	This expression substitutes for the smart type completion invocation at the variable position.
`componentTypeOf (<array variable or array type>)`	Returns component type of an array. For example, see the Live Templates in the iterations group in the other group.
`currentPackage()`	Returns the current package name.
`date(sDate)`	Returns the current system date in the specified format. By default, the current date is returned in the default system format. However, if you specify date format in double quotes, the date will be presented in this format: 
`decapitalize(sName)`	Replaces the first letter of the name passed as a parameter with the corresponding lowercase letter.
`descendantClassEnum(<String>)`	Shows the children of the class entered as a string parameter.
`enum(sCompletionString1,sCompletionString2,...)`	List of comma-delimited strings suggested for completion at the template invocation.

`escapeString(sEscapeString)`	Escapes the specified string.
`expectedType()`	Returns the type which is expected as a result of the whole template. Makes sense if the template is expanded in the right part of an assignment, after return, etc.
`fileName(sFileName)`	Returns file name with extension.
`fileNameWithoutExtension()`	Returns file name without extension.
`firstWord(sFirstWord)`	Returns the first word of the string passed as a parameter.
`groovyScript("groovy code")`	Returns Groovy script with the specified code. You can use `groovyScript` macro with multiple arguments. The first argument is a script text that is executed or a path to the file that contains a script. The next arguments are bound to `_1, _2, _3, ..._n` variables that are available inside your script. Also, `_editor` variable is available inside the script. This variable is bound to the current editor.
`guessElementType (<container>)`	Makes a guess on the type of elements stored in a `java.util.Collection`. To make a guess, IntelliJ IDEA tries to find the places where the elements were added to or extracted from the container.
`iterableComponentType(<ArrayOrIterable>)`	Returns the type of an iterable component, such as an array or a collection.
`iterableVariable()`	Returns the name of a variable that can be iterated.
`lineNumber()`	Returns the current line number.
`lowercaseAndDash(String)`	Returns lower case separated by dashes, of the string passed as a parameter. For example, the string `MyExampleName` is converted to `my-example-name`.
`methodName()`	Returns the name of the embracing method (where the template is expanded).
`methodParameters()`	Returns the list of parameters of the embracing method (where the template is expanded).
`methodReturnType()`	Returns the type of the value returned by the current method (the method within which the template is expanded).
`qualifiedClassName()`	Returns the fully qualified name of the current class (the class where the template is expanded). Tip Clear the Shorten FQ names check box.
`rightSideType()`	Declares the left-side variable with a type of the right-side expression. It is used in the iterations group to have a single template for generating both raw-type and Generics Collections.
`snakeCase(sCamelCaseText)`	Returns snake_case string out of CamelCase string passed as a parameter.
`spaceSeparated(String)`	Returns string separated with spaces out of CamelCase string passed as a parameter. For example, if the string passed as a parameter is `fooBar`, then the function returns `foo bar`.
`subtypes(sType)`	Returns the subtypes of the type passed as a parameter.
`suggestIndexName()`	Suggests the name of an index variable. Returns `i` if there is no such variable in scope, otherwise returns `j` if there is no such variable in scope, etc.
`suggestVariableName()`	Suggests the name for a variable based on the variable type and its initializer expression, according to your code style settings that refer to the variable naming rules. For example, if it is a variable that holds an element within iteration, IntelliJ IDEA makes a guess on the most reasonable names, also taking into account the name of the container being iterated.
`suggestFirstVariableName(sFirstVariableName)`	Doesn't suggest `true, false, this, super`.
`time(sSystemTime)`	Returns the current system time.
`typeOfVariable(VAR)`	Returns the type of the variable passed as a parameter.
`underscoresToCamelCase(sCamelCaseText)`	Returns the string passed as a parameter with CamelHump letters substituting for underscores. For example, if the string passed as a parameter is `foo_bar`, then the function returns `fooBar`.
`underscoresToSpaces(sParameterWithSpaces)`	Returns the string passed as a parameter with spaces substituting for underscores.
`user()`	Returns the name of the current user.
`variableOfType(<type>)`	Suggests all variables that may be assigned to the type passed as a parameter, for example variableOfType("java.util.Vector"). If you pass an empty string ("") as a parameter, suggests all variables regardless of their types.
`JsArrayVariable`	Returns JavaScript array name.
`jsClassName()`	Returns the name of the current JavaScript class.
`jsComponentType`	Returns the JavaScript component type.
`jsMethodName()`	Returns the name of the current JavaScript method.
`jsQualifiedClassName`	Returns the complete name of the current JavaScript class.
`jsSuggestIndexName`	Returns a suggested name for an index.
`jsSuggestVariableName`	Returns a suggested name for a variable.

Sharing Live Templates

On this page:

– Introduction
– Copying live templates
– Pasting live templates

Introduction

IntelliJ IDEA allows you to easily share the live templates with your team members, using the Copy and Paste commands on the context menu .

So doing, the Copy command creates a serialized XML representation of the selected live templates, and Paste inserts the templates based on this XML under the selected group.

This is how it's done.

Copying live templates

To create an XML copy of live templates

1. Open the Live Templates page of the Settings/Preferences dialog.
2. Select the desired templates. Use `Ctrl` and `Shift` keys for multiple selection.
3. Right-click the selection, and choose Copy on the context menu (or press `Ctrl+C`).
   An XML copy of the selected templates is created in the system clipboard.

Pasting live templates

To paste the live templates

1. In the Live Templates page of the Settings/Preferences dialog, select the group where you want to insert the copied live templates.
2. Right-click the selection, and choose Paste on the context menu (or press `Ctrl+V`).
   An XML copy of the selected templates is inserted from the system clipboard to the selected group of live templates.

Populating Projects

In this part:

- Populating Projects
  - Basics
- Creating Packages and package-info Files
- Creating Directories
- Creating Empty Files
- Creating Template-based Files
- Creating Java Classes, Interfaces, Enumerations and Annotations

Basics

Populate your project by creating new elements of various types (directories , packages and files). IntelliJ IDEA suggests the following alternative ways of accessing the corresponding functionality:

- File | New on the main menu.
- The New context menu command.
- The keyboard shortcut `Alt+Insert`.

As a result, the New menu is shown in which you select the type of an element to be created.

Creating Packages and package-info Files

On this page:

- Introduction
- Creating a package
- Creating a package-info.java file

Introduction

Within the folders marked as containing source code (see e.g. Folder categories), the New | Package and New | package-info.java commands are available for creating packages and package-info files.

Creating a package

To create a package, follow these steps:

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Select the destination folder or package (, or).
3. Do one of the following:
   - Select File | New | Package.
   - Select New | Package from the context menu.
   - Press `Alt+Insert` and select Package.
4. In the dialog that opens, specify the package name. This may be just a name (e.g. `helloworld`) or a sequence of names separated with dots (e.g. `com.example.helloworld`). In the latter case, the corresponding package structure will be created.

Creating a package-info.java file

To create a package-info.Java file, follow these steps:

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Select the package for which you want to create a package-info file.
3. Do one of the following:
   - Select File | New | package-info.java.
   - Select New | package-info.java from the context menu.
   - Press `Alt+Insert` and select package-info.java.

Tip You can edit the template for `package-info.java` in the Settings/Preferences dialog on the File and Code Templates page .

Creating Directories

Depending on the category of the folder you are currently in (see e.g. Folder categories), use the New | Directory or the New | Package command to create a directory or a number of nested directories.

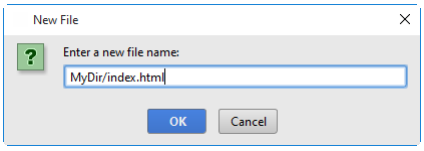
1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Select the destination directory.
3. Do one of the following:
   - Select File | New | Directory or File | New | Package depending on which option (Directory or Package) is available.
   - Select New | Directory or New | Package from the context menu.
   - Press `Alt+Insert` and select Directory or Package.
4. In the dialog that opens, specify the directory name. If you want to create a number of nested directories, specify the directory names separated with slashes (if you used the New | Directory command) or dots (if you used the New | Package command).

Creating Empty Files

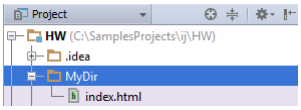
Generally, all the files that you create when developing applications are template-based . However, sometimes you may want to create empty files.

To create an empty file

1. Do one of the following:
   - In the Project tool window , select the directory or package in which you want to create a new file, and then choose File | New on the main menu.
   - Right-click the corresponding directory or package and select New from the context menu.
   - Press `Alt+Insert`
2. Select File.
3. In the New File dialog, in the field under Enter a new file name, type the file name and extension.
   You can specify the whole directory structure prepending the new file name. If the nested directories do not yet exist, they will be created:



4. Click OK.
   If the names of non-existent sub-directories were specified before the new file name, the whole structure will be created under the target directory:



5. If the extension you have specified is not associated with any of the file types recognized by IntelliJ IDEA, the Register New File Type Association dialog is displayed. In this dialog, you can associate the extension with one of the recognized file types. To do that, select the file type under Open matching files in IntelliJ IDEA and click OK. As a result, the extension is associated with the specified file type.

Tip If there are no appropriate file types for the new extension, you may want to create a new file type and associate the extension with that type. For more information, see Creating and Registering File Types .

Creating Template-based Files

On this page:

– Introduction

– Cresating a new file from a template

Introduction

IntelliJ IDEA provides file templates for most of the languages that it supports . This lets you create the files with the initial content appropriate for the file purpose. For example, there are individual file templates for creating Java classes, interfaces, enumerations, annotations, scripts, GUI forms, etc.

Generally, the file name extension for a template-based file is set automatically so you don't need to specify it. This, for example, is the case when creating Java types (classes, interfaces, etc.): the corresponding files always get the `.java` extension. Similarly, if you create a Groovy class, a new file will get the `.groovy` extension.

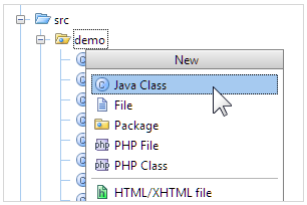
Sometimes, you are given an opportunity to select the desired extension from the list. This, for example, is the case when creating PHP classes.

Note that the set of file templates you can choose from, generally, depends on which facets your module has and, to a lesser extent, on your current location in the Project tool window .

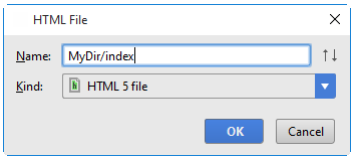
Cresating a new file from a template

To create a new template-based file

1. Do one of the following:
   – In the Project tool window , select the directory or package in which you want to create a new file, and then choose File | New on the main menu.
   – Right-click the corresponding directory or package and select New from the context menu.
   – Press `Alt+Insert`
2. Select the desired file type. Generally, all the options except File , Package and Directory correspond to using a file template.

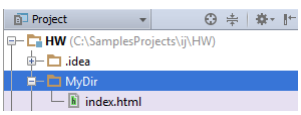


   An existing file template may be missing from the list for one of the following reasons:
   – The template is associated with a facet that your module doesn't have. In such a case, you may want to add the corresponding facet to your module.
   – The corresponding file type is inappropriate for the current location. For example, you may be trying to create a Java class outside of Java source or test directory which is not possible.
   – The template you are looking for is a custom template whose file name extension (template extension) does not match registered patterns of any of the recognized file types. In such a case, you may want to register the corresponding pattern for an existing recognized file type or add a new file type and register the corresponding pattern for this new type. For more information, see Creating and Registering File Types .
3. In the dialog that opens, type the name of the file in the corresponding field. Note that you should not type the file name extension.
   You can specify the whole directory structure prepending the new file name. If the nested directories do not yet exist, they will be created:



   If required, specify the kind of the new template-based file. For example, if you select Java class, you'll be able to create a Java class, interface, enumeration, annotation, etc.; if you select to create an HTML file, you'll be able to create HTML, HTML4, or XHTML file. In other words, use one of the corresponding related file templates.

   Specify other information as required. For example, you may be asked to define the values of custom variables if the corresponding file template contains such variables and their values are not currently set.
4. Click OK. The new file that corresponds to the selected file template will be created under the target location.
   If the names of non-existent sub-directories were specified before the new file name, the whole structure will be created under the target directory:



Sometimes, you may want to change the auto-generated file name extension. To do that, use the Rename refactoring (Refactor | Rename).

Creating Java Classes, Interfaces, Enumerations and Annotations

IntelliJ IDEA suggests a simplified way of creating Java classes, interfaces, enumerations or annotations, with all the required infrastructure. The stub generated from a file template is a file with the corresponding extension (`.java`). This file contains a package statement, a commented header section, and a class declaration.

To create Java class, interface, enumeration or annotation

1. In the Project tool window , select the destination package. Choose File | New or press `Alt+Insert` .
   Alternatively, right-click the package and select New from the context menu.
2. In the New menu, select Java Class.
3. In the Create New Class dialog:
   1. Type the name of the entity you are creating in the Name field. Don't type the file name extension.
      You can type the name of the package separated by dots in front of the entity you are creating. In this case, IntelliJ IDEA creates the appropriate package structure, generates a file with the specified name and the corresponding extension (`.java`), and opens this file in the editor.
   2. Select the kind of symbol to be created from the Kind list. Note that you can use the `Up` and `Down` arrow keys to browse the contents of this list.
   3. Click OK.

IntelliJ IDEA generates a file with the specified name and the corresponding extension (`.java`), and opens this file in the editor.

Generating Code

This section describes how to quickly and accurately populate your source code with the complicated code constructs.

In this part:

– Creating Code Constructs by Live Templates
– Creating Code Constructs Using Surround Templates
– Examples of Using Live Templates
– Generating Constructors
– Generating Delegation Methods
– Generating Equals() and HashCode()
– Generating Getters and Setters
– Generating ToString()
– Implementing Methods of an Interface
– Overriding Methods of a Superclass
– Rearranging Code Using Arrangement Rules
– Surrounding Blocks of Code with Language Constructs
– Unwrapping and Removing Statements
– Using Postfix Templates

Creating Code Constructs by Live Templates

On this page:

- Introduction
- Inserting a live template

Introduction

This page describes how to generate source code using live templates .

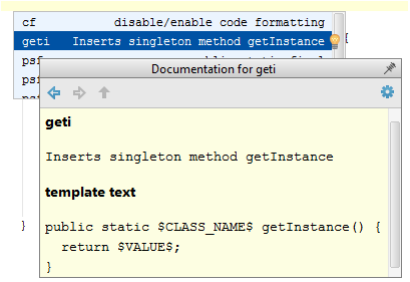
Using Live Templates enables you to create such code constructs as the main() method , iteration over an array , , typical variable declarations, output statements, Emmet, etc.

To explore the list of available live templates, in the Settings/Preferences dialog, open the Live Templates page.

Inserting a live template

To insert a live template in the source code

1. Place the caret at the desired position, where the new construct should be added.
2. Do one of the following:
   - On the main menu, choose Code | Insert Live Template.
   - Press `Ctrl+J`.
   - Type some initial letters of the template abbreviation to get the matching abbreviations in the suggestion list. Note that the suggestion list may contain same abbreviations for different templates.
3. From the suggestion list , select the desired template. While the suggestion list is displayed, it is possible to view Quick Documentation for the items at caret, by pressing `Ctrl+Q`.



4. Press the template invocation key (this may be `Space`, `Tab` or `Enter`, depending on the template definition). The new code construct is inserted in the specified position.
5. If the selected template is parametrized and requires user input , the editor enters the template editing mode and displays the first input field highlighted with the red frame. Type your value in this frame and press `Enter` or `Tab` to complete input and pass to the next input field. After completing the last input field, the caret moves to the end of the construct, and the editor returns to the regular mode of operation.

It is also possible to type a template abbreviation, and then press `Ctrl+J`.

Creating Code Constructs Using Surround Templates

This section describes how to wrap fragments of code according to surround templates .

To surround a block of code with a live template

1. In the editor, select the piece of code to be surrounded.
2. Do one of the following:
   - On the main menu, choose Code | Surround With Live Template....
   - Press `Ctrl+Alt+J`.
3. Select the desired template from the suggestion list.

Examples of Using Live Templates

This section contains examples demonstrating how to populate your source code with the complicated code constructs using live templates .

In this part:

- Generating main() Method. Example of Applying a Simple Live Template
- Iterating Over an Array. Example of Applying Parameterized Live Templates
- Wrapping a Tag. Example of Applying Surround Live Templates

Generating main() method. Example of Applying a Simple Live Template

As an example of applying a simple live template, let us create a main() method.

To create main() method

1. Open the desired class for editing.
2. Type the abbreviation of the main() method template:

```
psvm
```

3. Press the template invocation key. By default, it is `Tab`. The abbreviation expands to the `main()` method:

```
public static void main(String args[]) {

}
```

Iterating over an Array. Example of Applying Parameterized Live Templates

As an example of applying a parameterized template, let us create a construct that iterates over an array.

To iterate over an array

1. Open the desired class for editing.

2. Type the `itar` abbreviation:

```
int[]  arrayOfInts  = new int[]  {7,9,3,17,11,23,37};
char[] arrayOfChars = new char[] {'a', 'b', 'c'};
itar
```

3. Press the template invocation key. The abbreviation expands into a code construct, with the suggested values for the template's variables already inserted:

```
int[]  arrayOfInts  = new int[]  {7,9,3,17,11,23,37};
char[] arrayOfChars = new char[] {'a', 'b', 'c'};
for (int i = 0; i < arrayOfChars.length; i++) {
  char aChar = arrayOfChars[i];

}
```

4. The Editor is now in its special template editing mode. The cue is the red border around the index variable, which is the first input position in this template. IntelliJ IDEA automatically suggests `i` as the value for the index variable. If you want to change the suggested name, type a new one. All references to the variable within the expanded template will be automatically changed as you type the new name.

   In figure below, `i` is changed to `idx` :

```
int[]  arrayOfInts  = new int[]  {7,9,3,17,11,23,37};
char[] arrayOfChars = new char[] {'a', 'b', 'c'};
for (int idx = 0; idx < arrayOfChars.length; idx++) {
  char aChar = arrayOfChars[idx];

}
```

5. Press `Tab` or `Enter` to move to the next input position defined by the template:

```
int[]  arrayOfInts  = new int[]  {7,9,3,17,11,23,37};
char[] arrayOfChars = new char[] {'a', 'b', 'c'};
for (int idx = 0; idx < arrayOfChars.length; idx++) {
  char aChar = arr(
                   arrayOfChars   char[]
                   arrayOfInts    int[]
}
```

   As you can see on the image above, IntelliJ IDEA automatically detects all array elements in the scope, and suggests selecting the correct one. If there is only one array element in the scope, IntelliJ IDEA inserts its name automatically without displaying a pop-up list.

6. IntelliJ IDEA detects whether the type of the array elements changes. It automatically updates the type of the variable that holds the current array element, and suggests a list of reasonable names for it:

```
int[]  arrayOfInts  = new int[]  {7,9,3,17,11,23,37};
char[] arrayOfChars = new char[] {'a', 'b', 'c'};
for (int idx = 0; idx < arrayOfInts.length; idx++) {
  int anInt = arrayOfInts[idx];
```

```
anInt
arrayOfInt
i
ofInt
```

7. Press `Tab` or `Enter` to apply the selected/specified variant. This will move you to the next input position which in our case is the END position of the template:

```
int[]  arrayOfInts  = new int[]  {7,9,3,17,11,23,37};
char[] arrayOfChars = new char[] {'a', 'b', 'c'};
for (int idx = 0; idx < arrayOfInts.length; idx++) {
  int anInt = arrayOfInts[idx];

}
```

Wrapping a Tag. Example of Applying Surround Live Templates

As an example of applying a surround template, let's wrap a piece of XML code with tags.

To surround a code fragment

1. Open the desired file for editing.

2. Select a code fragment.

```
<company>
    <name>Peter</name>
    <name>John</name>
    <name>Sarah</name>
</company>
```

3. Press invocation shortcut `Ctrl+Alt+J`. IntelliJ IDEA suggests the following surround templates:

4. Select the tag template from the suggestion list. The code fragment is surrounded with empty tags:

```
<company>
    <>
        <name>Peter</name>
        <name>John</name>
        <name>Sarah</name>
    </>
</company>
```

5. The caret rests within the opening one. On typing the tag name in the opening tag, the name is automatically reproduced in the closing tag:

```
<company>
    <staff>
        <name>Peter</name>
        <name>John</name>
        <name>Sarah</name>
    </staff>
</company>
```

Generating Constructors

On this page:

– Introduction
– Generating a constructor
– Example

Introduction

Constructor generator makes it possible to create constructors with arguments. The value of these arguments are assigned to the field variables.

The generated constructors are inserted at the points defined in the Order of Members section of the Code Style settings. By default, the code generator places constructors after the fields.

Generating a constructor

To generate a constructor

1. On the main menu, choose Code | Generate. Alternatively, right-click the editor and choose Generate on the context menu, or use `Alt+Insert` keyboard shortcut.
2. In the pop-up list that is displayed in the editor, select Constructor option.
3. If the class in question contains fields, IntelliJ IDEA suggests you to select the fields to be initialized by constructor. In the Choose Fields to Initialize by Constructor dialog, select the desired fields.

   Tip Use `Ctrl` and `Shift` keys for multiple selection.
4. Click OK.

Example

```
public class MyClass
                    {
                    int aInteger;
                    double bDouble;

                    public MyClass(int myAIntegerParam, double myBDoubleParam) {
                    aInteger = myAIntegerParam;
                    bDouble = myBDoubleParam;
                    }
                    }
```

Generating Delegation Methods

On this page:

– Introduction
– Creating delegation methods
– Example

Introduction

You can create methods that delegate behavior to the fields or methods of your class. This approach makes it possible to give access to the information of a field or method without direct exposing this field or method.

Creating delegation methods

To create a delegation method

1. Do one of the following:
   – On the main menu, choose Code | Generate.
   – Right-click the editor and choose Generate on the context menu
   – Press `Alt+Insert`.
2. In the pop-up list that is displayed in the editor, select Delegate Methods. Select Target To Generate Delegates For dialog appears, displaying the list of objects to be delegated to.
3. Select the target field or method, and click OK. The Select Method To Generate Delegation For dialog appears, displaying the list of methods to delegate.
4. Select the desired methods. For multiple selection, use `Ctrl` and `Shift` keys. Click OK.

Example

`Currency` class has a field `calendar` of the type `Calendar`. To gain access to certain functionality of the `Calendar` class from the `Currency` class, we need to create a new method that will delegate the request to `calendar`.

```
Calendar calendar;
                    public int get(int
                    i) {
                    return calendar.get(i);
                    }
```

Generating equals() and hashCode()

On this page:

- Basics
- Generating equals() and hashCode() methods
- Example

Basics

```
public boolean equals(Object obj)
```

This method returns `true`, if an object passed to it as an argument is equal to the object on which this method is invoked.

```
public int hashCode()
```

This method returns the integer hash code value for the object on which this method is invoked.

Generating equals() and hashCode() methods

Tip If `equals()` and `hashCode()` methods already exist in a class, you will be prompted to delete them before proceeding.

To generate `equals()` and `hashCode()` methods, follow these steps:

1. Do one of the following:
   - On the main menu, choose Code | Generate.
   - Right-click the editor and choose Generate on the context menu.
   - Press `Alt+Insert`.
2. From the pop-up list that opens in the editor, select equals() and hashCode() option to open the Generate equals() and hashCode() wizard .
3. Follow the steps of the wizard:
   - On the first page of the wizard, select or clear the check boxes to accept subclasses, and use getters during code generation. You can also select a velocity template from the Template drop-down list to generate the code or create a custom template.
   - On the second page of the wizard, select the fields that should be used to determine equality, and click Next.
   - On the third page, select the fields to generate hash code.
     Note that only the fields that were included in the `equals()` method, can participate in creating hash code. All these fields are selected by default, but you can unselect them, if necessary.

     Click Next.
   - On the fourth page of the wizard, select the fields that contain non-null values. This optional li help the generated code avoid check for null and thus improves performance. Click Finish.

Example

```
public boolean equals(Object o) {
                    if (this == o) return true;
                    if (o == null
                         || getClass() != o.getClass()) return false;
                    FixedRateCurrencyExchangeService that = (FixedRateCurrencyExchangeService) o;
                    if (Double.compare(that.rate,
                    rate) != 0)
                    return false;
                    return true;
                    }

                    public int hashCode() {
                    long temp = rate !=
                    +0.0d ? Double.doubleToLongBits(rate) :
                    0L;
                    return int (temp ^
                    (temp >>> 32));
                    }
```

Generating Getters and Setters

On this page:

– Introduction
– Generating accessor and mutator methods
– Example 1
– Example 2

Introduction

You can generate accessor and mutator methods (getters and setters) for the fields that exist in your classes. IntelliJ IDEA generates getters and setters with only one argument, as required by the JavaBeans API.

The getter and setter method names are generated by IntelliJ IDEA according your code generation naming preferences .

Generating accessor and mutator methods

To create accessor and mutator methods

1. On the main menu, choose Code | Generate.

   Alternatively, right-click the editor and choose Generate on the context menu, or use `Alt+Insert` keyboard shortcut.

2. In the pop-up list that is displayed in the editor, select one of the following options:
   Getter
   Accessor method for getting the current value of the selected fields.
   Setter
   Mutator method for setting specified values to the selected fields.
   Getter and Setter
   Both methods for the selected fields.

3. In the Choose Fields to Generate Getters and Setters dialog box, select the desired fields.
   You can add a custom setter or getter by clicking … and accessing Getter/Setter Templates dialog.

   If getters and setters for a field already exist, this field is not included in the list.

4. Click OK.

Example 1

Consider the following code:

```
public class MyClass
                    {
                    int aInteger;
                    }
```

In the Naming section of the Code Generation page, parameter prefix is set to `my`, and parameter suffix to `Param`.

After generating the getter and setter the following code will be produced:

```
public class MyClass
                    {
                    int aInteger;
                    public int getAInteger() {
                    return aInteger;
                    }
                    public void setAInteger (int myAIntegerParam) {
                    aInteger = myAIntegerParam;
                    }
                    }
```

Example 2

However, if `a` is specified as a field prefix in the Code Generation page, then it will not take part in the generation of the method and parameter names:

```
public class MyClass
                    {
                    int aInteger;
                    public int getInteger() {
                    return Integer;
                    }
                    public void setInteger (int myIntegerParam) {
                    aInteger = myIntegerParam;
                    }
                    }
```

Generating toString()

On this page:

- Introduction
- Generating toString() method
  - Inspections
  - Logging
- Examples
  - Basic code
  - Getter is enabled
  - Excluding fields and methods
  - JavaDoc

Introduction

The action Generate toString() enables creating or updating `toString()` method.

The beans usually needs to dump their field values for debug purposes, and it's rather tedious to write the dump code for it. The Generate toString() action generates the `toString()` method, dumping all the fields in a simple manner.

Generating toString() method

To generate toString() method

1. Open the desired class for editing and do one of the following:
   - On the main menu, choose Code | Generate.
   - Right-click the editor and choose Generate on the context menu
   - Press `Alt+Insert`.
2. From the pop-up list that shows up, select toString() option. Generate toString() wizard displays the list of fields in the class.
3. In the wizard, specify the following:
   - Select the fields to be used to generate a `toString()` method.
     By default, all the available fields are selected. Clicking the button Select None results in adding a `toString()` method consisting of method declaration and return statement only.
   - Select the desired way of generating a `toString()` method from the Templates drop-down list.
   - Select the check box Insert @Override if necessary.
     Refer to the section Generate ToString() Dialog for details.
   - If you are not happy with the settings, click the Settings button. This results in showing the toString() Generation Settings dialog box, where one can tune the function's behavior. Refer to the section Generate ToString() Settings Dialog for details.

   When ready, click OK.

Tip If `toString()` method already exists in a class, you will be prompted to delete this method before proceeding, depending on the value of When method already exists setting (Generate ToString() Settings Dialog).

Inspections

There are two related code inspections under the node toString() issues:

- Class does not override 'toString()' method
- Field not used in 'toString()' method

The inspection Class does not override toString() method can be used to identify any classes where you might have forgotten to add a `toString()` method.

This inspection uses the exclude settings to ignore classes with the fields not supposed to be dumped. An additional settings is to exclude certain classes by using a regular expression matching their classname. As default, this is used to exclude any Exception classes.

The inspection Field not used in 'toString()' method: This inspection can be used to identify out-of-synchronization situations, where you have an existing `toString()` method that dumps the fields. However, some fields have been added to the class later, and these new fields are not dumped in the `toString()` method.

Change the severity of this inspection by enabling it to show errors as warnings. This will highlight any unused fields on-the-fly in the editor; the right gutter will indicate the location of the errors using a yellow marker.

Logging

Log4j is used for logging. To enable logging, open for editing the file `log.xml` used by IntelliJ IDEA. This file resides in the `bin` folder of IntelliJ IDEA installation. Add the following lines to this file:

```
<category name="org.jetbrains.generate.tostring">
<priority value="DEBUG"/>
<appender-ref ref="FILE"/>
</category>
```

Examples

Basic code

Consider the following code:

```
public class MyServerConfigBean
                    {
                    private String name;
                    private String url;
                    private int port;
                    private String[] validIPs;
                    ...
                    }
```

Place the caret somewhere within the class, press `Alt+Insert`, and choose toString() from the pop-up list. The following method is now added to the bean:

```
public String toString() {
                    return "MyServerConfigBean{" +
                    "name='" + name +
                    '\'' +
                    ", url='" + url +
                    '\'' +
                    ", port=" + port +
                    ", validIps=" + Arrays.toString(validIps) +
                    '}';
                    }
```

Getter is enabled

Consider the following code:

```
public class MyServerConfigBean
                    {
                    private String name;
                    private String url;
                    private int port;
                    private String[] validIPs;
                    ...
                    public String getServerAddress()
                    {
                    return url + ":" + port;
                    }
                    ...
                    }
```

Place the caret somewhere within the class, press `Alt+Insert`, and choose toString() from the pop-up list. After invoking the action Generate toString(), the result is:

```
public String toString() {
                    return "MyServerConfigBean{" +
                    "name='" + name +
                    '\'' +
                    ", url='" + url +
                    '\'' +
                    ", port=" + port +
                    ", serverAddress='" + getServerAddress() + "'" +
                    ", validIps=" + Arrays.toString(validIps) +
                    '}'
                    }
```

Excluding fields and methods

Refer to the Exclude section of the Settings dialog.

Usually you don't want to add constant fields as the debug information in your `toString()` method. So you select the check box Exclude constant fields, and prevent constant fields in the output.

Besides that, it is possible to filter by the field's name, to exclude, say, an internal debug field. So, type `^debug` in the text field Exclude fields by name (reg exp), to prevent debug fields.

The example below shows the results with excluded fields. The original code is:

```
public class MyServerConfigBean
                    {
                    private final static String
                    USERNAME = "scott";
                    private final static String
                    PASSWORD = "tiger";
                    private String name;
                    private String url;
                    private int port;
                    private String[] validIPs;
                    ...
                    public String getServerAddress()
                    {
                    return url + ":" + port;
                    private boolean debug = true;
                    }
                    ...
                    }
```

After generating a `toString()` method, the code looks like the following:

```
public String toString() {
                    return "MyServerConfigBean{" +
                    "name='" + name +
                    '\'' +
                    ", url='" + url +
                    '\'' +
                    ", port=" + port +
                    ", validIps=" + Arrays.toString(validIps) +
                    '}'
                    }
```

As you see, the constant fields (USERNAME, PASSWORD) are not used in the generated code. The regular expression excludes the debug field. The excluded fields do not appear in the Generate ToString() Dialog .

To exclude a method, select the check box Exclude methods by name (reg exp) field. For example, if you type `^getCausedBy.*` in the text field Exclude methods by name (reg exp), you will thus prevent outputting methods whose names start with `getCausedBy` .

JavaDoc

It is possible to add JavaDoc comments to the generated `toString()` method. This is done by inserting the JavaDoc comments in a Velocity template . See the following template example:

```
/**
                    * Insert your JavaDoc comments here
                    *
                    * @return a string representation of the object.
                    */
                    return "$classname{}";
```

Implementing Methods of an Interface

On this page:

– Introduction
– Implementing methods
– Changing method body

Introduction

If a class is declared as implementing a certain interface or extending a class with abstract methods, it has to implement the methods of such interface or class. IntelliJ IDEA creates stubs of the implemented methods, with the default return values for the primitive types, and null values for the objects.

Implementing methods

To implement method of an interface or abstract class , follow these steps:

1. Do one of the following:
   – On the main menu, choose Code | Implement method.
   – Press `Ctrl+I`
   – Right-click the editor, choose Generate on the context menu, or press `Alt+Insert`, and choose Implement methods.

   The Select methods to implement dialog appears, displaying the list of classes and interfaces with the methods that can be implemented.
2. Select one or more methods to implement. For multiple selection, use `Ctrl` and `Shift` keys.
3. If necessary, select the check box Copy JavaDoc to insert JavaDoc comments from the implemented interface of abstract methods (if any).
4. Click OK.

Changing method body

File template responsible for implementing a method (Implemented method body) accepts predefined template variables from "File Header" File | Settings/IntelliJ IDEA | Preferences - Editor - File and Code Templates - Code - File Header e.g. `${USER}` , `${DATE}` , etc.

For example, consider the following file template:

```
#if ( $RETURN_TYPE
                    !=
                    "void" )return $DEFAULT_RETURN_VALUE;#end
                    // TODO ($USER, $DATE):To change the body of an implemented method, use File | Settings - Editor -  File
                    and Code Templates.
```

Provided that an implemented interface contains two methods, this template expands into the following code:

```
@Override
                    public void hunt() {
                    // TODO (wombat, 9/21/12): To change the body of an implemented method, use File | Settings - Editor -  File
                    and Code Templates.
                    }
                    @Override
                    public String sniff() {
                    return null;  // TODO (wombat, 9/21/12):
                    To change body of implemented methods use File | Settings - Editor - File and Code Templates.
                    }
```

Overriding Methods of a Superclass

On this page:

- Overview
- Overriding methods
- Changing method body

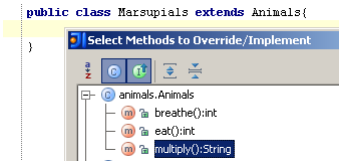
Overview

You can override any method of a parent class, using the code generation facility. IntelliJ IDEA creates a stub that contains a call to the method of the superclass, leaving the developer with the task of providing some meaningful source code.

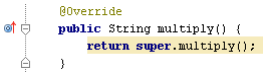
Overriding methods

To override methods, follow these steps:

1. With the class in question having the focus, invoke the Override method command in one of the following ways:
   - Press `Ctrl+O`.
   - On the main menu, choose Code | Override method.
   - Right-click the editor, choose Generate on the context menu, or press `Alt+Insert`, and choose Override methods.
2. Select methods that can be overridden from the Select methods to override dialog box. The list of methods does not include the methods that are already overridden, or cannot be accessed from the current subclass.



3. Select one or more methods to override.
4. If necessary, select the following options:
   - Insert @Override to add the `@Override` annotation.
   - Copy JavaDoc to insert JavaDoc comments from the overridden methods (if any).
5. Having generated the overriding method, create the required source code. Note the icon that marks the overriding method in the left gutter. Use this icon to view the name of the base class, and navigate to the overridden method .



Changing method body

File template responsible for overriding a method (Overridden method body) accepts predefined template variables from "File Header" (File | Settings - Editor - File and Code Templates - Code - File Header), e.g. `${USER}`, `${DATE}`, etc.

For example, consider the following file template:

```
#if ( $RETURN_TYPE
                    !=
                    "void" )return $DEFAULT_RETURN_VALUE;#end  // TODO ($USER, $DATE):To change
                    the method body, use Settings - Editor - File and Code Templates.
```

Provided that an overridden class contains two methods, this template expands into the following code:

```
public void breathe() {
                    // TODO (wombat, 9/21/12): To change the method body, use Settings - Editor - File and Code Templates.
                    }
                    public void eat() {
                    // TODO (wombat, 9/21/12): To change  the method body, use Settings - Editor - File and Code Templates.
                    }
```

Rearranging Code Using Arrangement Rules

On this page:

– Defining Arrangement Rules

– Rearranging Code

– Example

Defining Arrangement Rules

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Code Style under Editor.
2. Under the Code Style node, select the appropriate language.
3. On the Arrangement tab of the language settings, specify rules for your code rearrangement and click OK.

Rearranging Code

You can rearrange your code according to your arrangement rules preferences .

1. In the editor select a part of the code that you want to rearrange.
2. On the main menu, choose Code | Rearrange Code.

   Alternatively, in the project tool window, right-click the file where you want to perform the code rearranging, select Code | Reformat Code.
3. In the dialog that opens, select Rearrange entries check box and click OK.

Example

Consider the following code that lets you test an arrangement rule that puts fields before methods:

Before After

```
class Test {
                    public void  test(){
                    }
                    private int  i;
                    }
                    class Test2 {
                    public void test(){
                    }
                    private int i;
                    private int j;
                    }
```

```
class Test {
                    private int i;
                    public void test() {
                    }
                    }
                    class Test2 {
                    private int  i;
                    private int  j;
                    public void test() {
                    }
                    }
```

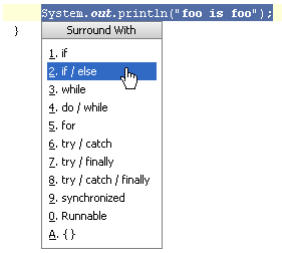
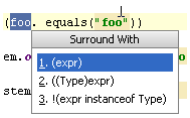
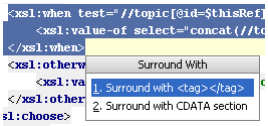
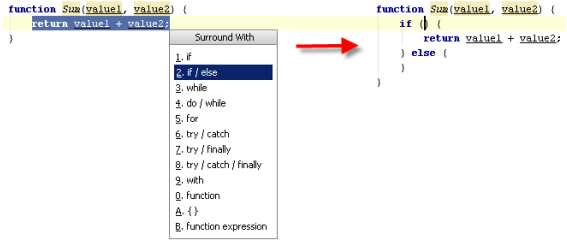
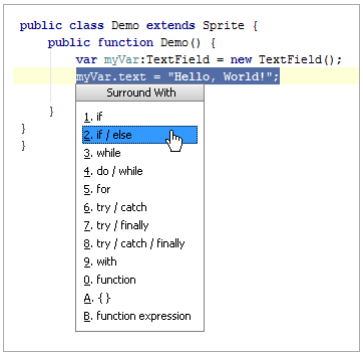
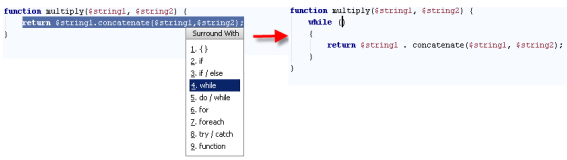
Surrounding Blocks of Code with Language Constructs

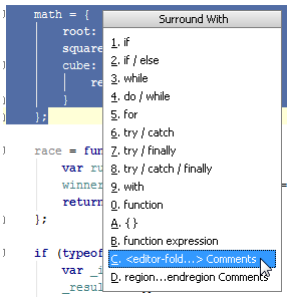
On this page:

- Applicable contexts
- Surrounding blocks of code

Applicable contexts

The Surround with feature allows you easily put expressions or statements within blocks or language constructs. This feature in IntelliJ IDEA applies to:

Context	Surround with	Example
Java statements	– `if` – `if/else` – `while` – `do/while` – `for` – `try/catch` – `try/finally` – `try/catch/finally` – `synchronized` – `Runnable` – `{}`	
Java expressions	– `(expr)` – `((Type)expr)` – `!(expr instanseof Type)`	
XML/HTML /JSP/JSPX tags	– Tag – CDATA section – <% ... %> – Emmet	
JavaScript statements	– `(expr)` – `!(expr)` – `if` – `if / else` – `while` – `do / while` – `for` – `try / catch` – `try / finally` – `try / catch / finally` – `with` – `function` – `{ }` – `function expression`	
ActionScript statements	– `if` – `if / else` – `while` – `do / while` – `for` – `try / catch` – `try / finally` – `try / catch / finally` – `with` – `function` – `{}` – `function expression`	
PHP statements	– `if` – `if / else` – `while` – `do / while` – `for` – `foreach` – `try / catch` – `function`	
Custom folding region comments	Any fragment of code, where Surround With is applicable	



Surrounding blocks of code

To surround a block of code

1. Select the desired code fragment.
2. Do one of the following:
   - On the main menu, choose Code | Surround With
   - Press `Ctrl+Alt+T`.

   A pop-up window displays the list of enclosing statements according to the context.
3. Select the desired surround statement from the list. To do that, use the mouse cursor, up and down arrow keys, or a shortcut key displayed next to each element of the list.

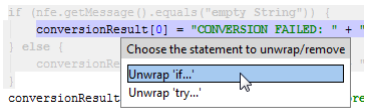
Unwrapping and Removing Statements

IntelliJ IDEA enables you to quickly unwrap or extract expressions from the enclosing statements. This action is available for:

- Java: `for, foreach, if..elseif...else, try...catch...finally, while...do, do...while`, and lone braces.
- Groovy
- JavaScript
- XML, HTML, and JSP

To unwrap or remove a statement

1. Place the caret on the expression you want to extract or unwrap.
2. Choose Code | Unwrap/Remove on the main menu or press `Ctrl+Shift+Delete`. IntelliJ IDEA shows a pop-up window with all the actions that are available in the current context. Statements to be extracted are displayed on the blue background, statements to be removed are displayed on the grey background.



3. Click the desired action or select it using the up and down arrow keys and press `Enter`.

Using Postfix Templates

On this page:

– Overview
– Activating and de-activating postfix templates
– Using postfix templates

Overview

Postfix code completion lets you transform an already typed expression to another one based on the postfix you type after a dot, the type of the expression, and its context. This transformation is performed by expanding the postfix-specific predefined template. For example, the `.if` postfix applied to a boolean expression wraps it with an `if` statement. See more at: Postfix Code Completion .

Activating and de-activating postfix templates

When the postfix template functionality is enabled, IntelliJ IDEA transforms expressions with postfixes into other expressions by expanding the corresponding postfix-specific templates.

To activate the postfix templates functionality:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click Postfix under General.
   The Postfix Completion page that opens shows a list of postfixes for which IntelliJ IDEA can transform expressions according to postfix-specific templates. Transformation templates are shown next to corresponding postfixes. When you select a postfix, the Description pane illustrates the corresponding transformation showing the code snippet before and after the template is expanded.
2. Select the Enable postfix templates check box and choose the key to launch template expansion from the Expand template with drop-down list.
3. Select the check boxes next to the postfixes for which you want IntelliJ IDEA to expand the corresponding templates.

To suppress expanding a template, clear the check box nex to the corresponding postfix. To block the postfix templates functionality, clear the Enable postfix templates check box.

Using postfix templates

To apply a postfix template:

1. Type your code followed by a dot, until a suggestion list appears, or press `Ctrl+J` .
2. Choose the desired postfix template from the suggestion list. If necessary, choose an expression to be surrounded or replaced.
3. Press the expansion key, defined in the Postfix Completion page of the editor settings.

Auto-Completing Code

This part describes the various techniques that enable you to speed up the editing process. These techniques include:

- Basic code completion on `Ctrl+Space`.
- Type completion on `Ctrl+Shift+Space`.
- Completing punctuation on `Enter`.
- Completing statements with smart `Enter`.
- Completing paths in the Select Path dialog .
- Expanding words with `Alt+Slash`.
- Negating expressions with an exclamation mark.
- Various tips and tricks on using the suggestion lists that appear on invoking code completion.

Tip All types of code completion offered by IntelliJ IDEA support "CamelHumps" and "Middle Matching". It means that you can either complete any item by entering only its uppercase characters, or start typing any part of an item.

All completion results are ranged and ordered by relevance considering the caret position and surrounding code.

Basic Code Completion. Completing Names and Keywords

On this page:

– Basics
– Completing names
– Tips and tricks

Basics

Basic code completion helps you complete the names of classes, methods, fields, and keywords within the visibility scope. When you invoke code completion, IntelliJ IDEA analyses the context and suggests the choices that are reachable from the current position of the caret.

If basic code completion is applied to a part of a field, parameter, or variable declaration, IntelliJ IDEA suggests a list of possible names with regard to the type of the item.

Invoking Basic code completion for the second time shows the names of classes available through the module dependencies.

When invoked once more (for the third time in a row), the suggestion list expands to the whole project, regardless of the dependencies.

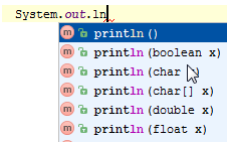
This action completes the names of classes and interfaces , searching through the entire project. If the desired class is not yet imported, it will be imported automatically.

Code completion covers supported file types and custom file types. However, IntelliJ IDEA does not recognize the structure of custom file types and suggests completion options regardless of whether a specific type is appropriate in the current context.

Completing names

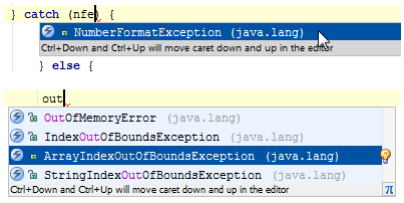
To automatically complete names in the current visibility scope, follow these general steps:

1. Type a name or a part of a name.
   While typing, use the following techniques:
   – Narrow down the suggestion list by typing any part of a word, even the characters somewhere in the middle, or invoking code completion after a dot separator. IntelliJ IDEA will show suggestion list that includes occurrences with all the possible positions of the entered characters.
   This makes the use of the asterisk wildcard unnecessary:

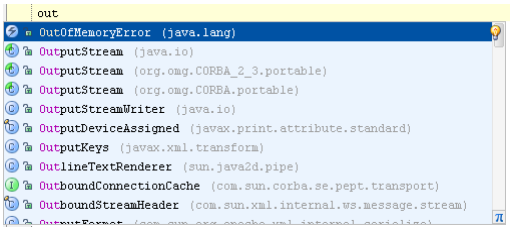


   – In case of CamelCase or snake_case names, type the initial letters only. So doing, IntelliJ IDEA automatically recognizes and matches the initial letters.

2. Do one of the following:
   – On the main menu, choose Code | Completion | Basic.
   – Press `Ctrl+Space` .



3. If necessary, press `Ctrl+Space` for the second time.
   This shows all classes, available through the module dependencies. The second completion shows also static fields and methods.



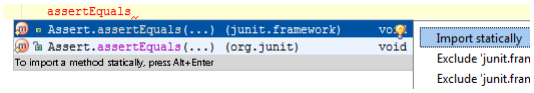
   Tip This action produces the same effect as pressing `Ctrl+Alt+Space` .

   When invoked for the third time, basic code completion expands the suggestion list to all classes throughout the project, regardless of the dependencies.

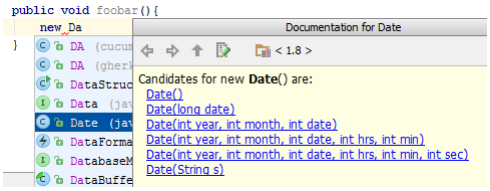
4. Select the desired class from the suggestion list, and do one of the following:
   – Press `Enter` or double-click the desired choice .
   – Press `Tab` .

Tips and tricks

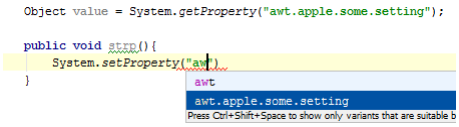
– IntelliJ IDEA makes it possible to choose between adding `Class.methodName()` at the insertion point, and adding a static import statement. In the first variant, just complete selection .
  In the second variant, press `Alt+Enter` in the suggestion list, and choose Import statically.



- Pressing `Ctrl+Q` in a list of arguments shows documentation popup for constructors as well. If a class has more than one constructor, the links are shown:



- If a suggestion list shows n variants of a class, type a dot, and the next suggestion list after dot will show all the possible qualified names of the members. After selecting the desired one, import statement will be added automatically.
- Suggestion list includes literals, already defined in a file.



Smart Type Code Completion. Completing Code Based on Type Information

On this page:

– Introduction

– Using smart type code completion

Introduction

Smart Type code completion filters the suggestion list and includes only those types that are applicable to the current context.

Thus, you can use smart type completion in the situations, when it is possible to determine the appropriate type:

– In the right part of the assignment statements.

– In variable initializers.

– In `return` statements.

– In the list of parameters of a method call.

– After the `new` keyword in an object declaration.

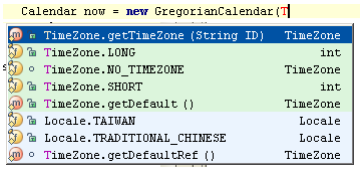
Using smart type code completion

To use smart type code completion

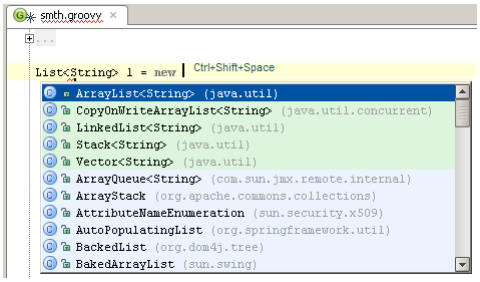
1. Do one of the following:
   – On the main menu, choose Code | Complete Code | SmartType.
   – Press `Ctrl+Shift+Space`.

   SmartType code completion automatically highlights selection in the suggestion list that seems most suitable for the current context. Such results display on a darker green background.

   For example, consider smart type code completion after the `new` keyword in Java:

   or in Groovy:

   As you type, the suggestion list reduces to show matching entries only.

2. If necessary, press `Ctrl+Shift+Space` once more.
   Thus you can complete, for example:
   – Chained expressions.
   – Collections, lists and arrays. IntelliJ IDEA searches for the symbols with the same component type and suggest to convert them.
   – Static method calls or constant references. IntelliJ IDEA scans for the static methods and fields, but suggests only the ones with the type suitable in the current context.

Completing Statements

On this page:

- Completing statements
- Examples
  - Completing a method declaration
  - Completing code constructs
  - Automatic encapsulation

Completing statements

The Smart Enter completion enables you to create syntactically correct code constructs.

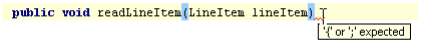
To automatically complete a statement, follow these steps:

- Start typing a statement, and press `Ctrl+Shift+Enter`.
  The punctuation required in the current context is added and the caret moves to the next editing position.

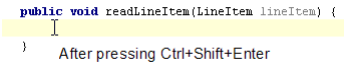
Examples

Completing a method declaration

Start typing a method declaration, and press `Ctrl+Shift+Enter` after the opening parenthesis:



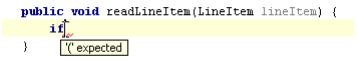
This will create the entire construct of a method, the caret resting inside the method body:



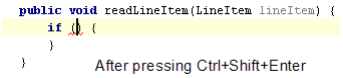
After pressing Ctrl+Shift+Enter

Completing code constructs

Start typing a code construct:



After pressing `Ctrl+Shift+Enter`, the construct is completed, and the caret is placed at the next editing position:



After pressing Ctrl+Shift+Enter

Automatic encapsulation

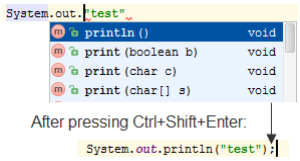
IntelliJ IDEA automatically encapsulates a method call, when you directly type a new method call next to it. For example, consider the following code:

```
|"test"
```

Note the caret that rests just before the opening quote. Then type the method call.

When `println` gets the focus in the suggestion list, select it with `Ctrl+Shift+Enter`.

```
System.out.println("test");
```



Completing Path

On this page:

– Introduction

– Completing path

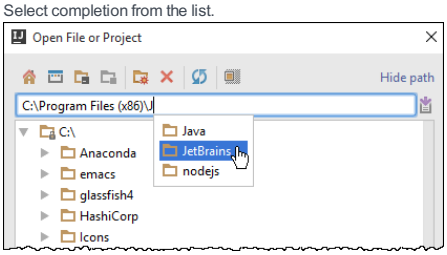
Introduction

This type of completion aims at speeding up selection of files, folders, etc. The dialog used for that purpose is called rather often, for example when adding a new SDK in the Project Structure dialog , or specifying an application server home directory. .

Completing path

To automatically complete a path

1. Open the Select Path dialog by doing one of the following:
   – Click the Browse button .
   – Click New on the Project SDK page of the Project Structure dialog box.
2. Make sure that the Path field is visible. If the field is hidden, click the Show path link.
3. Start typing a path. IntelliJ IDEA suggests a drop-down list where you can quickly find the needed directory. As you type, the suggestion list shrinks to show the matching paths only. You can also press `Ctrl+Space` to show the suggestion list.
4. Select completion from the list.



Completing Punctuation

On this page:

- Introduction
- Configuring paired elements behavior

Introduction

In the Editor , you can configure the editor's behavior related to the paired braces. When the option Insert pair } is enabled, the closing brace is automatically added with indent on pressing `Enter` . Indentation level is defined in the Code Style settings .

Configuring paired elements behavior

To configure insertion of paired elements

1. Open the Settings dialog box , and under the Editor node, click Smart Keys .
2. Configure automatic insertion of paired elements by selecting the following check boxes:
   - Insert pair bracket
   - Insert pair quote
   - Auto-close tag on typing '</'
   - Insert pair %> in JSP
3. To have the closing braces inserted upon pressing `Enter` , select the Insert pair } check box.
4. Apply changes and click OK.

Completing Tag Names

On this page:

– Introduction
– Completing tag names
– Inserting taglib declaration
– Importing a taglib declaration

Introduction

IntelliJ IDEA automatically completes the names of tags, attributes and their values in files of the following types:

– HTML/XHTML
– XML/XSL
– JSP/JSPX

Automatic tag name completion is based on the DTD or Schema the file is associated with.

If there is no schema association, IntelliJ IDEA will use the file content (tag and attribute names and their values) to complete your input.

In XML/XSL and JSP/JSPX files, completion for taglibs and namespaces is available.

You can have IntelliJ IDEA do one of the following:

– Insert a declaration of the taglib the tag in question belongs to.
– Import the desired taglib and insert all required import and reference statements.

Completing tag names

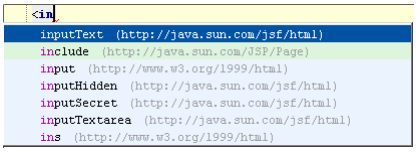
To invoke the tag name completion

1. Press `<` and start typing the tag name. IntelliJ IDEA displays the list of tag names appropriate in the current context. Use the `ArrowUp` and `ArrowDown` buttons to scroll through the list.
2. Press `Enter` to accept selection from the list, or continue typing the tag name manually.
   IntelliJ IDEA automatically inserts mandatory attributes according to the schema.

Inserting taglib declaration

To insert a taglib declaration

1. Start typing the tag and press `Ctrl+Alt+Space`.
2. From the list of matching tags, select the desired one. The `uri` of the taglib it belongs to is displayed in brackets.

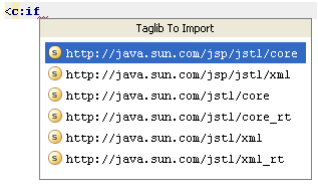


3. Select the desired taglib and press `Enter`. IntelliJ IDEA adds the declaration of the selected taglib.

Importing a taglib declaration

To import a taglib

1. Start typing the taglib prefix and press `Alt+Insert`.
2. Select the desired taglib from the list of suggestions and press `Enter`.



IntelliJ IDEA imports the selected taglib and adds the import statement automatically.

Hippie Completion. Cyclic Expanding Words

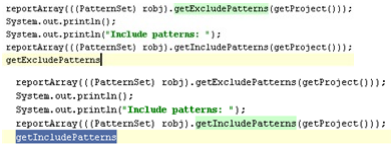
This helpful feature analyses your text in the visible scope and suggests to complete a word with a keyword, class name, method or variable. So doing, the prototype is highlighted in the source code.

To expand a string at caret to an existing word

1. Type the initial string.
2. Do one of the following:
   - On the main menu, choose Code | Completion | Cyclic Expand Word.
   - Press `Alt+Slash`.

   The editor shows the first suggested value.
3. Press `Enter` to accept the suggested word, or continue pressing `Alt+Slash` until the desired word is found.

Example:

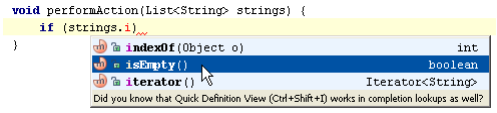


Negating Expressions

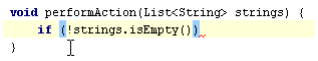
When you use code completion in Java, you can finish selection from the suggestion list with an exclamation mark. So doing, the exclamation mark will be placed before a Boolean method or variable being completed, thus negating the whole expression.

To negate an expression

1. Start typing an expression, and press `Ctrl+Space`.
2. In the suggestion list, select the desired part of the expression.



3. Press `!` key. Your expression is completed with negation:



Using Suggestion List

On this page:

- Introduction
- Navigating through the suggestion lists
- Narrowing and widening the suggestion list
- Completing selection
- Viewing reference information in the suggestion lists
- Sorting entries in a suggestion list
- Viewing hierarchy in a completion pop-up
- Closing a suggestion list

Introduction

Any code completion or live template operation ends up with a suggestion list, from which you have to select the desired option.

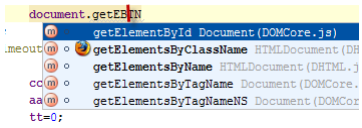
IntelliJ IDEA provides handy ways to make your selection, complete the source code, sort entries, and view reference information without leaving the suggestion list.

Navigating through the suggestion lists

To navigate through the suggestion lists, use the mouse cursor or the `Up` and `Down` arrow keys.

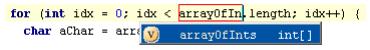
Narrowing and widening the suggestion list

- Use the horizontal arrow keys in the completed expression to make the suggestion list wider (left arrow) or closer (right arrow):

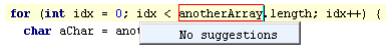


Completing selection

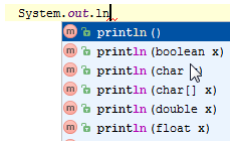
- Use `Enter` to insert the selected string at the insertion point.
- Use `Tab` to replace the string next to the caret with the selected one.
- Use `Ctrl+Shift+Enter` to make the current code construct syntactically correct (balance parentheses, add missing braces and semicolons, etc.)
- It is possible to enter the desired element name manually when the input position is highlighted and the list of suggestions is opened. As you type, the suggestion list shrinks so that only matching element names remain:



If you enter a name not used in scope, IntelliJ IDEA informs you with a pop-up window, as shown in the figure below:



- It is possible to type any part of a word, even the characters located somewhere in the middle:



That's why asterisk character doesn't get any special processing as a wildcard.

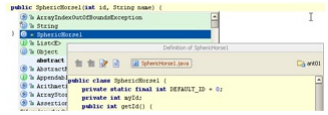
- Use `!` to negate a Boolean expression.

Please note the following:

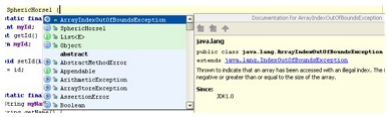
- If, invoking an action, you don't get what you want immediately, invoke this action again by pressing the same shortcut .
- Most suitable variants in the suggestion list are displayed on top of the list.
- Entries in bold font on top of the list denote symbols defined in the current class. All the other symbols are shown below.

Viewing reference information in the suggestion lists

- Use the Quick Definition View . If you select an entry in a list and press `Ctrl+Shift+I`, IntelliJ IDEA displays the following quick information pop-up window:



- Quick Information View works in a suggestion list. If you select an entry in a list, and press `Ctrl+Q`, IntelliJ IDEA displays the following quick information pop-up window:



Sorting entries in a suggestion list

To sort entries in a suggestion list:

- Click A or π in the lower-right corner of the list to toggle between sorting in alphabetical order, or by relevance.

Click [A] or [#] in the lower right corner of the list to toggle between sorting in alphabetical order, or by relevance.

Please note the following:

- The sorting icon appears in the list if it is long enough. For few entries only sorting icon is not displayed.
- Use mouse only to click this icon.
- IntelliJ IDEA memorizes the type of sorting.
- Default behavior is defined in the Code Completion page of the Editor settings.

Viewing hierarchy in a completion pop-up

To view hierarchy in a completion pop-up, follow these steps:

1. Start typing the source code, and press `Ctrl+Shift+Space` or `Ctrl+Space`. The suggestion list appears.
2. While in the suggestion list, press `Ctrl+H`. The Hierarchy tool window shows the type hierarchy of the class, selected in the suggestion list.
   Same way, for the current selection in the suggestion list, you can view call hierarchy (`Ctrl+Alt+H`), or method hierarchy (`Ctrl+Shift+H`).

Closing a suggestion list

Close suggestion lists with `Escape`, then enter your own code at the code completion position.

Creating and Optimizing Imports

In this part:

- Creating Imports
- Excluding Classes from Auto-Import
- Optimizing Imports

Creating Imports

On this page:

- Introduction
- Importing packages on the fly
- Completing a short class name and importing a PHP namespace on-the-fly
- Importing a PHP namespace using a quick fix
- Importing an XML namespace
- Importing TypeScript symbols

Introduction

When you reference a class that has not been imported, IntelliJ IDEA helps you locate this file and add it to the list of imports . You can import a single class or an entire package, depending on your settings.

The import statement adds to the imports section, but the cursor does not move from the current position, and your current editing session does not interrupt. This feature is known as the Import Assistant.

Tip To configure the way auto-import works, open the IntelliJ IDEA settings , and then go to the Editor: Auto Import page.

The same possibility applies to the XML , JSP, and JSPX files. When you type a tag with an unbound namespace , import assistant suggests to create a namespace and offers a list of appropriate choices.

When you reference a PHP class that is defined outside the current file, IntelliJ IDEA locates the class definition, whereupon you can do one of the following:

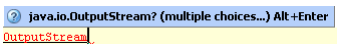
- Have IntelliJ IDEA automatically complete the fully qualified class name, including the namespace the class is defined in. This will result in littering your code.
- Have IntelliJ IDEA automatically complete the short class name and import the namespace the class is defined in.
- Import the namespace manually using a quick fix.

In the PHP context, a namespace is imported by inserting a `use` statement.

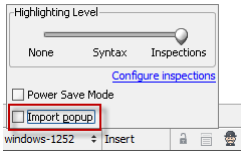
Importing packages on the fly

To import packages on-the-fly, follow these steps:

1. Start typing a name in the editor. If the name references a class that has not been imported, the following prompt appears:

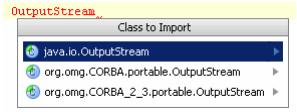
   

   If the pop-up annoys you, change this behavior for the current file. Just click Hector in the Status bar , and clear the check box Import Pop-up:

   The unresolved references will be underlined, and you will have to invoke intention action Add import explicitly.

2. Press `Alt+Enter` . If there are multiple choices, select the desired import from the list.

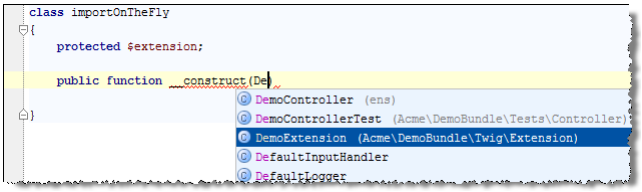
   

Tip If the option Add unambiguous imports on the fly is checked, IntelliJ IDEA does not inform you about the needed import, if there is only one choice, and adds the only possible import automatically.

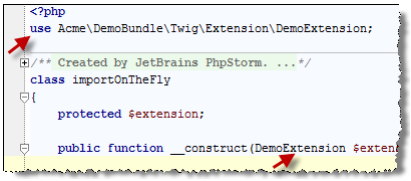
Completing a short class name and importing a PHP namespace on-the-fly

To compile a short class name and import a PHP namespace, follow these steps:

1. To enable on-the-fly namespace import, open the IntelliJ IDEA settings , and then click Auto Import under the Editor node. In the Editor: Auto Import page that opens, configure automatic namespace import in the PHP section.
   - To have IntelliJ IDEA automatically import PHP namespaces, add use statements, and complete short class names on the fly when you are typing in a class or file that belongs to a certain namespace, select the Enable auto-import in namespace scope check box. This check box is selected by default.
   - To have automatic namespace import applied when you are typing in a file that does not belong to any specific namespace, select the Enable auto-import in file scope check box.
2. Open the desired file for editing and start typing the short name of a class.
3. From the suggested variants for completion , select the relevant class name:

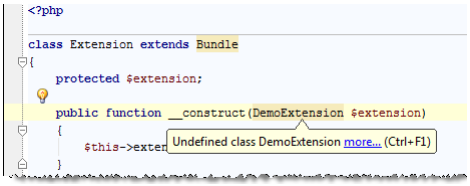
   IntelliJ IDEA completes the short class name and inserts a `use` statement with the namespace where the selected class is defined:



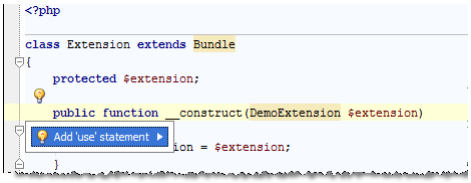
Importing a PHP namespace using a quick fix

To import a PHP class using a quick fix, follow these steps:

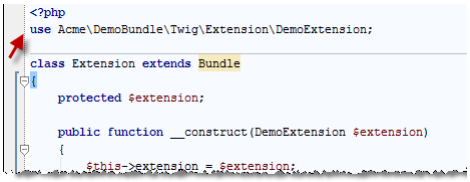
1. Open the desired file for editing and reference a PHP class. If the referenced class is not bound, the following prompt appears:



2. Press `Alt+Enter`. IntelliJ IDEA suggests to import the namespace where it has detected the declaration of the class:



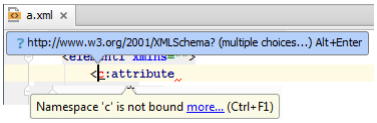
3. Press Enter. IntelliJ IDEA inserts a namespace declaration statement (`use` statement):



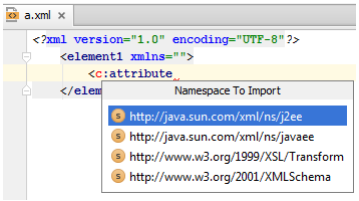
Importing an XML namespace

To import an XML namespace, follow these steps:

1. Open the desired file for editing, and start typing a tag. If a namespace is not bound, the following prompt appears:



2. Press `Alt+Enter`. If there are multiple choices, select the desired namespace from the list.

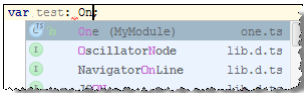


Depending on the file type, IntelliJ IDEA creates a namespace declaration, or a taglib:

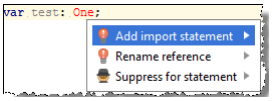
Importing TypeScript symbols

In the TypeScript context, IntelliJ IDEA can generate `import` statements for modules, classes, and any other symbol that can be exported and called as a type. Open the desired file in the editor and do one of the following:

- Start typing the short name of a symbol. From the suggested variants for completion, select the relevant symbol name:



- Position the cursor at the unresolved symbol, which is displayed in red, and press `Alt+Enter`:

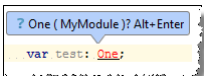


On the context menu, select Add import statement and press `Enter`.

- Configure IntelliJ IDEA to show a pop-up every time you hover the mouse pointer over an unresolved reference which required import:
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click Auto Import under General.

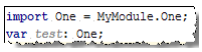
2. On the Auto Import page that opens, select the Show import pop-up check box in the TypeScript area.

Every time you hover the mouse pointer over an unresolved symbol, IntelliJ IDEA will display the following pop-up message:



Press `Alt+Enter` to have an import statement generated and inserted automatically.

In either case, IntelliJ IDEA inserts an `import` statement:



Excluding Classes from Auto-Import

On this page:

– Introduction
– Configuring imports
– Excluding classes from imports

Introduction

The list of imports suggested by IntelliJ IDEA can be sometimes far too wide, and can include the classes you don't actually need. For example, some of the class names in your project can match the names of internal SDK or unrelated library classes.

IntelliJ IDEA lets you configure the classes to be excluded from import assistance and code completion, thus helping avoid unintentional use of the wrong classes. Besides that, IntelliJ IDEA provides an intention action that allows you to exclude unnecessary classes on-the-fly.

These settings apply to code completion as well. The classes and packages specified as the ones to be ignored by the code completion feature, will not be added to the suggestion list.

Configuring imports

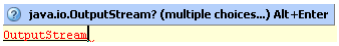
To configure classes to be excluded from import

1. Open the Settings/Preferences dialog box , and under the Editor / General node, click Auto-Import.
2. On the Auto-Import page, click + (`Alt+Insert`).
3. In the dialog box that opens, type the name of the class or a whole package to be excluded, and click OK.
   Use + (`Alt+Insert`) and – (`Alt+Delete`) to manage the list of classes and packages that IntelliJ IDEA shouldn't place to the suggestion list.
4. Apply changes and close Settings/Preferences dialog.

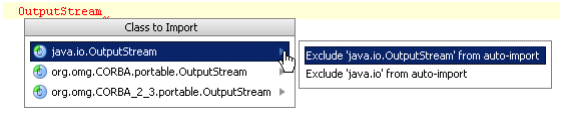
Excluding classes from imports

To exclude classes from imports in the fly

1. Start typing a name in the editor, and use intention action as described in the section Creating Imports :



2. In the Class to Import suggestion list, select the class you want to be ignored, and click the right arrow to reveal the nested list of intention actions. IntelliJ IDEA suggests you to exclude specific class or the whole containing package:



3. Click the desired class or package to be excluded.

Optimizing Imports

On this page:

- Introduction
- Optimizing imports in project
- Optimizing imports in the current file

Introduction

Sooner or later, some of the imported classes or packages become redundant to the code.

Typically, you have to stop what you are doing, scroll to the head of the file, find the unused imports, and remove them. It is rather easy to forget to remove imports when you remove usages.

IntelliJ IDEA provides the Optimize Imports feature, which enables you, whenever it is convenient, to remove unused imports from your current file, or from all files in the current directory at once. This helps you avoid unused, excessive and duplicating imports in your project.

One can remove unused import statements in the entire project or in the current file only.

Optimizing imports in project

To optimize imports in the entire project, follow these steps:

1. Place the caret at the Project tool window and do one of the following:
   - On the main menu, choose Code | Optimize Imports.
   - On the context menus of the Project tool window, choose Optimize Imports.
   - Press `Ctrl+Alt+O`.

   The Optimize Imports dialog box opens.
2. If your project is under version control, the option Only VCS changed files is enabled. Select or clear this option as required.
3. Click Run.

Optimizing imports in the current file

One way of dealing with unused import is to use the quick-fix that appears when you set the caret at the highlighted unused import . However, you can optimize imports in a larger scope as described below.

To optimize imports in the currently opened file, do one of the following:

- On the main menu, choose Code | Optimize Imports.
- Press `Ctrl+Alt+O`.
- Place the caret at the import statements and press `Alt+Enter`, or just click 💡 to show the list of suggested intention actions, and then choose Optimize imports.

Analyzing Applications

In this part:

- Dependencies Analysis
- Analyzing Dependencies Using DSM
- Viewing Structure and Hierarchy of the Source Code
- Analyzing Backward Dependencies
- Analyzing Cyclic Dependencies
- Analyzing Dependencies
- Analyzing Duplicates
- Analyzing Module Dependencies
- Analyzing External Stacktraces
- Analyzing Data Flow
- Validating Dependencies

Dependencies Analysis

IntelliJ IDEA suggests the following means of analyzing dependencies in your projects:

- Analysis of usages, which helps you locate all references to a certain class, variable, method or parameter. This facility includes search and view usages across the project and highlighting usages in a file.
- Possibility to view file structure .
- Possibility to explore hierarchy of the types, methods, and method calls.
- Search for repetitive code fragments .
- Analysis of dependencies (module , backward , or cyclic).
- Exploring complicated dependencies using Dependency Structure Matrix Analysis .

Analyzing Dependencies Using DSM

In this section:

- DSM Analysis
- Accessing DSM Analysis
- Expanding Dependencies
- Exploring Dependencies
- Find Usages for Dependencies
- Limiting DSM Scope

Tip Make sure that DSM Analysis plugin is installed and enabled. Refer to the Plugins page of the Settings dialog.

DSM Analysis

While working on complicated projects that definitely have numerous dependencies, you might experience difficulties trying to understand where to look for the problems. You can perform dependencies analysis, but you have to know exactly which dependencies you would like to analyze. This is where DSM helps.

DSM stands for Dependency Structure Matrix - a method for exploring dependencies between program parts (modules, classes, etc.), and provides a compact matrix representation of a project.

DSM analysis helps you visualize the dependencies between the parts of a project (modules, classes etc.) and highlights the information flow within a project.

DSM analysis can be used to manage how changes will affect the project. For example, if one of the classes needs to be changed, you can identify all dependencies and see how the change will propagate through the project. A dependency structure matrix lists all of the parts of a project and dependencies between them.

Results of the DSM analysis display in the special DSM view .

Tip This functionality is only available when the DSM Analysis plugin is enabled in your IntelliJ IDEA installation.

To invoke DSM

1. On the main menu, choose Analyze | Analyze Dependency Matrix. Note that this command becomes available on the menu if only the DSM Analysis plugin has been enabled. Refer to the Plugins page of the Settings dialog.

2. In the Specify Analyze Dependency Matrix Scope dialog, click the radio button that corresponds to the desired analysis scope.

3. Check the Include test sources check box, if you want to analyze the test sources. Click OK.

   Warning! If your project class files are out-of-date, the analysis may result in incomplete or incorrect data. To avoid this, IntelliJ IDEA asks you whether you want to compile a project before continuing the DSM analysis.

4. When ready, the DSM View is opened in a new window, enabling you to examine dependencies.

Expanding Dependencies

You can expand dependencies to examine them in more detail.

To expand dependencies

1. In the DSM view, select the cell to explore.
2. Double-click the cell. This will expand both rows displaying dependencies in more detail.

Exploring Dependencies

There is a possibility to limit view to selected dependencies only. Note, that in contrary to Limit Scope, only classes which produce dependencies selected are left.

To explore dependencies

1. In the DSM view, select the cell to explore.
2. Right-click the dependency and choose Explore Dependencies Between on the context menu. The classes that produce these dependencies will be opened in a new tab within the DSM view.

Find Usages for Dependencies

You can open selected dependencies in Find Usages view for further source-code analysis.

To find usages for dependencies

1. In the DSM view, select the dependency.
2. Right-click the dependency and choose Find Usages for Dependencies on the context menu.

Limiting DSM Scope

You can limit the scope of your DSM to the selected rows. Only these will remain in the new matrix.

To limit DSM scope

1. In the DSM view, select the rows to remain.
2. Right-click the selected rows and choose Limit Scope To Selection in the menu. The limited scope will be opened in a new tab within the DSM view.

Viewing Structure and Hierarchy of the Source Code

IntelliJ IDEA enables you to examine the hierarchy of classes, methods, and calls in the Hierarchy tool window, and explore the structure of source files in the Structure tool window.

- Both the Hierarchy and the Structure tool windows are available from the View menu.
- the Hierarchy tool window only becomes available, when a hierarchy is built.
- Hierarchies are built in the Navigate menu.

In this part:

- Building Call Hierarchy
- Building Class Hierarchy
- Building Method Hierarchy
- Retaining Hierarchy Tabs
- Viewing Hierarchies
- Viewing Structure of a Source File

Building Call Hierarchy

You can build and view the hierarchy of callers and callees for a selected method in the Hierarchy tool window . Before viewing call hierarchy, you need to build at least one.

To build a hierarchy of method calls

1. In the Editor, place the caret at the method declaration or usage. In the Project view, or another tool window, select the desired method.

2. Do one of the following:
   - On the main menu, choose Navigate | Call Hierarchy.
   - Press `Ctrl+Alt+H`.

Building Class Hierarchy

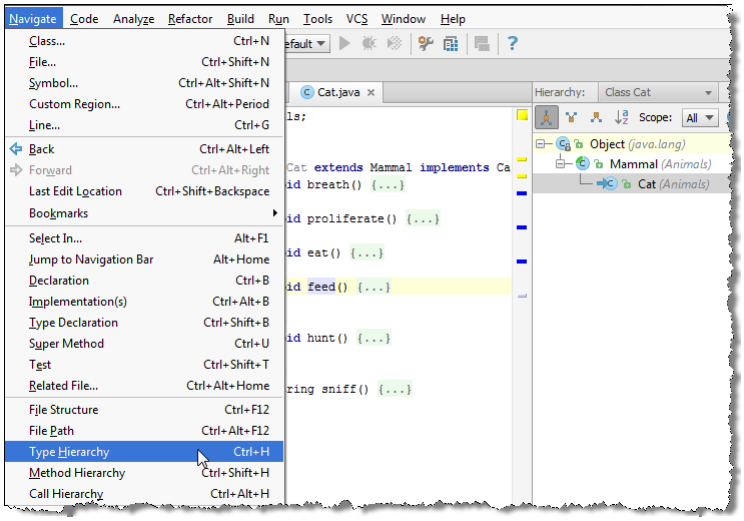
You can explore the hierarchy of parent and children classes of a selected class in the Hierarchy tool window .

Note that the Hierarchy tool window only becomes available, when you build the class hierarchy, as described below.

To build the hierarchy of classes

1. Select the desired class in the Project tool window, or open it in the editor.
2. On the main menu, choose Navigate | Type Hierarchy or press `Ctrl+H` .

The class hierarchy appears in the Hierarchy tool window .



Building Method Hierarchy

A method hierarchy makes it possible to examine a tree-view of the classes where a given method is:

- defined (+).
- not defined, but defined in the superclass (-).
- to be defined, because the class is not abstract (!).

To build a method hierarchy

1. Select the desired method in the Project tool window, or place the caret at the method declaration in the editor.
2. Do one of the following:
   - On the main menu, choose Navigate | Method Hierarchy.
   - Press `Ctrl+Shift+H`.

Retaining Hierarchy Tabs

By default, every time you build a new hierarchy, IntelliJ IDEA overwrites the current tab in the Hierarchy tool window. You can retain the contents of the desired tabs and have next hierarchies built in new tabs.

To retain a hierarchy tab

- In the Hierarchy tool window, click the Pin Tab button on the toolbar.

Viewing Hierarchies

Once built , hierarchies can be brought up for close examination in the Hierarchy tool window .

On this page:

– Showing the Hierarchy tool window
– Navigating between the tabs of the Hierarchy tool window
– Toggling between views

To show the Hierarchy tool window, do one of following

Warning! The Hierarchy tool window is not shown when there are no hierarchies to display. You have to build hierarchies first.

Refer to Building Class Hierarchy , Building Call Hierarchy , and Building Method Hierarchy to learn how to build hierarchies.

– On the main menu, choose View | Tool Windows | Hierarchy.

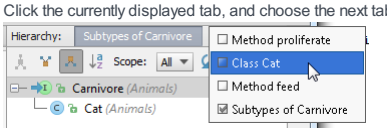
– Use `Alt+8` keyboard shortcut.

To navigate between the tabs of the Hierarchy window, do one of the following

– Right-click the currently displayed tab, and choose Select Next Tab/ Select Previous Tab on the context menu.

– Use the `Alt+Right` and `Alt+Left` keyboard shortcuts.

– Click the currently displayed tab, and choose the next tab to display.



To toggle between views, use the toolbar of the Hierarchy tool window

Toggling between views means showing ascending or descending hierarchy (callee vs. caller methods, parent vs. children classes etc.)

– Click to show caller methods, or supertypes.

– Click to show callee methods, or subtypes.

Viewing Structure of a Source File

On this page:

– Basics

– Viewing the structure of a file

– Viewing members

Basics

You can examine the structure of the file currently opened in the editor using the Structure tool window or the Structure pop-up window.

By default, IntelliJ IDEA shows all the classes, methods, etc. presented in the current file.

To have other members displayed , click the corresponding buttons on the toolbar of the Structure tool window.

You can also have class members shown in the Project tool window.

Viewing the structure of a file

To view the file structure, do one of the following

– On the main menu, choose View | Tool Windows | Structure to show the Structure tool window .

– Press Structure tool button to show the Structure tool window .

– Press `Alt+7` to show the Structure tool window .

– Press `Ctrl+F12` to show the Structure popup .

Viewing members

To have class fields displayed

– Click on the toolbar of the Structure tool window.

To have inherited members displayed

– Click on the toolbar of the Structure tool window.

By default, IntelliJ IDEA shows only methods, constants, and fields defined in the current class. If shown, inherited members are displayed gray.

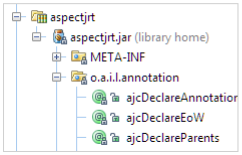
To have included files displayed

– Click on the toolbar.

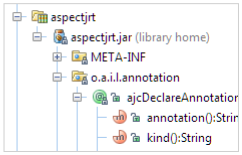
To have class members shown in the Project tool window

– Turn on the Show members item on the context menu of the Project tool window title bar. If this option is on, the files in the tree that contain classes turn into nodes. When such node is unfolded, the contained classes with their fields, methods, and other members of the selected item are shown.

Show Members: OFF



Show Members: ON



Analyzing Backward Dependencies

With this type of analysis you can find another classes or modules in a certain scope of interest, that depend on the specified scope of analysis (a whole project, a module, a file, unversioned files etc.). Results of the analysis display in a dedicated tab of the Dependency Viewer .

Backward dependencies analysis might be considerably time-consuming, especially on large projects.

To analyze a project for backward dependencies

1. On the main menu, choose Analyze | Analyze Backward Dependencies. The Specify Backward Dependency Analysis Scope dialog box opens.
2. In the Analysis Scope section, specify the part of your project, for which you would like to find the dependencies.
3. In the Scope of Interest section, specify the scope where the dependencies are sought for. You can select one of the pre-defined scopes from the drop-down list, or click the ellipsis button and create your own scope in the Scopes dialog .
4. Select the Include test sources check box, if you wish to analyze the test sources.
5. Click OK to run analysis. Productivity hints are displayed while the analysis is in progress. When ready, the Dependency Viewer opens a special tab, enabling you to examine dependencies.
6. In the left pane of the Dependency Viewer, select the node to be sought for. In the right pane select the scope to find usages of the selected node. Search results display in the lower pane of the tab.

Analyzing Cyclic Dependencies

Cyclic dependencies analysis enables you to detect any circular relationships between the packages in the specified scope. Results of analysis display in a dedicated tab of the Dependency Viewer .

To analyze a project for cyclic dependencies

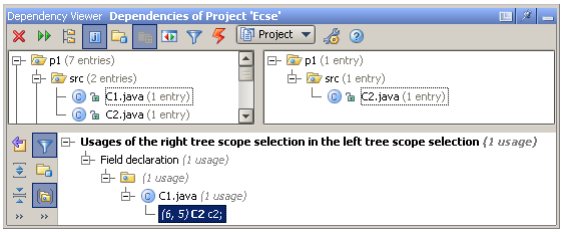
1. On the main menu, choose Analyze | Analyze Cyclic Dependencies.
2. In the Specify Cyclic Dependency Analysis Scope dialog , select the desired scope of analysis.
3. Click OK to run the analysis. Productivity hints are displayed while the analysis is in progress. When ready, the Dependency Viewer opens a special tab, enabling you to examine the dependencies.
4. In the left pane of the Dependency Viewer, select the node to be sought for. In the right pane select the scope to find usages of the selected node. The search results display in the lower pane of the tab.

Analyzing Dependencies

IntelliJ IDEA enables you to analyze the source code of your project and detect the dependencies, in which your application participates. The results of each dependencies analysis display in a separate tab of the Dependencies Viewer .

To analyze the dependencies in your project

1. On the main menu, choose Analyze | Analyze Dependencies. Alternatively, right-click the element you want to analyze - a package, a class etc. - and choose the same command on the context menu in the Project tool window , or in the editor.
2. In the Specify Dependency Analysis Scope dialog box, select the desired scope of analysis.
3. Examine dependencies in the Dependencies Viewer .



Analyzing Duplicates

On this page:

- Overview
- Searching for duplicates
- Detecting duplicates on-the-fly

Overview

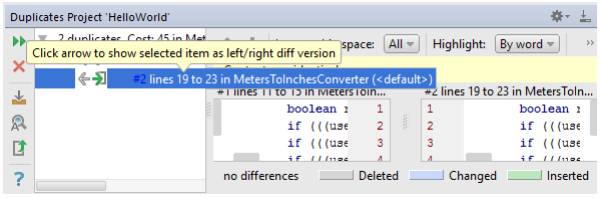
IntelliJ IDEA helps you find repetitive blocks of code in a certain range. This range can be a single file, a project, a module, or a custom scope. Results of analysis display in the dedicated tab of the Duplicates tool window .

Searching for duplicates

To search for duplicates

1. Do one of the following:
   - On the main menu, choose Analyze | Locate Duplicates.
   - Choose the Analyze | Locate Duplicates command on the context menu of the editor, or Project tool window .
2. In the Specify Code Duplication Analysis Scope dialog , specify the analysis scope (whole project, current file, uncommitted files (for the projects under version control), or some custom scope). In addition, you can include test sources into the analysis too.
   Click OK, when ready.
3. In the Code Duplication Analysis Settings dialog , do the following:
   1. Select languages to perform analysis in.
   2. For each language, check the options to define your preferences for the analysis.
      For example, you can opt to request identical match for code fragments to be considered duplicates, or specify a certain limit below which the code constructs are not considered duplicates (to avoid reporting about each `if` construct in the source code).

   Click OK.
4. In the Duplicates tool window , explore search results.

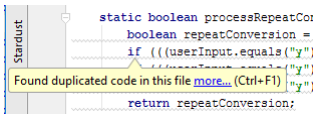


   - View the list of duplicates in the left pane of the tool window.
   - View differences between the found duplicates in the right pane. Use the arrow buttons to place the selected duplicate in one of the sections of the differences viewer and compare fragments of the code.
   - Navigate to the duplicates in the editor, using Jump to Source or Show Source commands of the duplicates context menu.
   - Eliminate duplicates from the source code by clicking and specifying the method name and parameters in the Extract Method dialog. This procedure is similar to the Extract method refactoring , with the only difference that in case of duplicates analysis the repetitive blocks of code are found automatically.

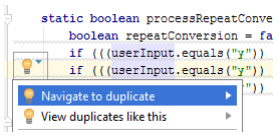
Detecting duplicates on-the-fly

IntelliJ IDEA enables spotting duplicates on-the-fly. This is done by the inspection General | Duplicated Code.

If you stumble upon an existing duplicate, or somehow create one either by writing or pasting code, you will know it instantly:

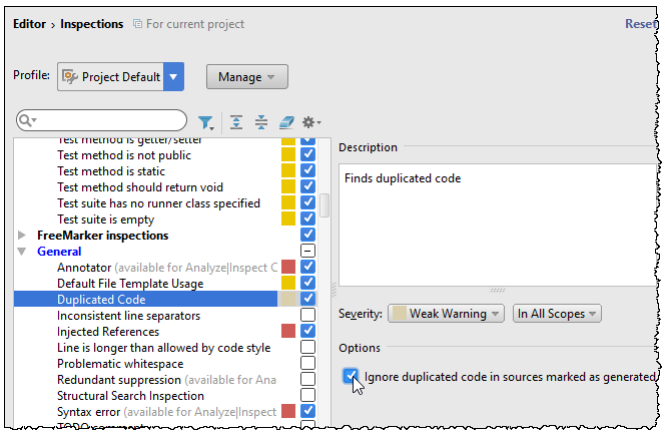


Inspection is accompanied by quick fixes, which enable you to navigate to the detected duplicates, or view all of them in the Find tool window :



Note that IntelliJ IDEA helps avoid locating duplicates in the generated sources .

To do that, select the check box Ignore duplicated code in sources marked as generated in the inspection settings page:


Editor › Inspections
For current project
Reset
Profile:
Project Default
Manage
Test method is getter/setter
Test method is not public
Test method is static
Test method should return void
Test suite has no runner class specified
Test suite is empty
FreeMarker inspections
General
Annotator (available for Analyze|Inspect C
Default File Template Usage
Duplicated Code
Inconsistent line separators
Injected References
Line is longer than allowed by code style
Problematic whitespace
Redundant suppression (available for Ana
Structural Search Inspection
Syntax error (available for Analyze|Inspect
Description
Finds duplicated code
Severity:
Weak Warning
In All Scopes
Options
Ignore duplicated code in sources marked as generated

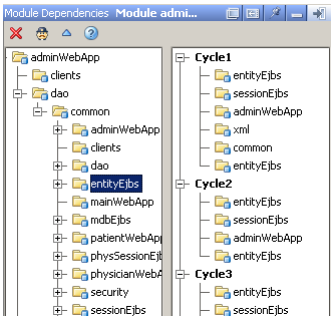
Analyzing Module Dependencies

Module dependencies analysis shows all modules that exist in the specified scope, the relationships between these modules, as they are specified in the Dependencies tab of the Project Structure dialog , and the cyclic dependencies between the modules.

Tip You might want to use this type of analysis to make sure that the dependencies you have previously defined, still exist in your project.

To analyze module dependencies

1. On the main menu, choose Analyze | Analyze Module Dependencies.
2. Specify the scope of analysis. You can opt to select the whole project, or a specific module.
3. Examine dependencies in the Module Dependencies Tool Window



4. Select a module in the tree view, and use the toolbar button of the Module Dependencies tool window to find the modules that depend on the selected one .

Analyzing External Stacktraces

On this page:

– Overview

– Analyzing external stacktrace

Overview

You might want to analyze exceptions received by someone else, for example, QA engineers, or investigate a deadlock, or a hang-problem. Unlike the exceptions that you get in the debug mode or when running unit tests, these exceptions do not have links that help you navigate to the corresponding locations in the source code. Moreover, the source code can be scrambled.

With IntelliJ IDEA, you can simply copy an exception or full thread dump, paste it to the Stacktrace Analyzer, explore information, and navigate to the corresponding source code.

Analyzing external stacktrace

To analyze an external stack trace or thread dump

1. On the main menu, choose Analyze | Analyze Stacktrace.
2. In the Analyze Stacktrace dialog box that opens, paste the external stack trace or thread dump into the Put a stacktrace or a complete thread dump here: text area.
3. Specify whether you want to have the stacktrace unscrambled. To do that, select the Unscramble stacktrace check box, select the desired unscrambler and log file.
4. If the stacktrace text is corrupted (lines are cut or wrapped, or too long, etc.) after processing with some software (for example, a bug tracker or a mail client), click Normalize.
5. Click OK. The stacktrace is displayed in the Run tool window.

Tip In case of thread dump, IntelliJ IDEA will present all threads in a readable way and sort them to show first those most probably responsible for the deadlock or hang-problem in question.

Analyzing Data Flow

On this page:

- Introduction
- Analyzing data flow
- Dataflow analysis results
  - Dataflow to Here
  - Dataflow from Here

Introduction

IntelliJ IDEA provides Dataflow Analysis features to help you with code archeology - better understand the inherited project code, interpret complicated parts of the code, find bottlenecks in the source, and more.

Specifically, Dataflow to/from here features allow you to:

- See where do values assigned to a variable come from.
- Find out what are all possible values the variable can have.
- Find out what are the places this expression\variable\method parameter can flow into.
- Reveal places where potential `NullPointerException` can arise.

If you want to trace back the value passed to a parameter at caret, you can create a slice view of the source code using the Analyze | Dataflow to Here command. Moreover, using Analyze | Dataflow from Here command you can find the places an expression can flow into. The results of each dataflow analysis are displayed in the dedicated tabs of the Analyze Dataflow tool window.

Analyzing data flow

To analyze dataflow to/from a symbol

1. Open the desired file for editing.
2. Place the caret at the symbol to be analyzed (expression\variable\method parameter).
3. On the main menu, or on the context menu, choose Analyze | Dataflow to Here or Analyze | Dataflow from Here depending on your purposes.
4. Specify the analysis scope and select, whether you want to ignore all values coming from the test code.
5. Click OK. Review the analysis results in the dedicated Analyze Dataflow tool window.

Dataflow analysis results

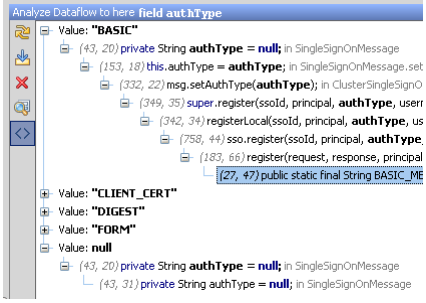
The following sections briefly describe how to "read" the dataflow analysis results.

Dataflow to Here

The image below shows an example of the dataflow to here analysis results.

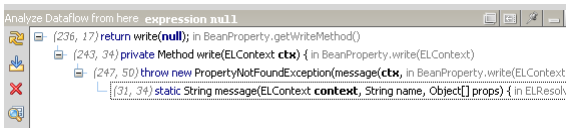
- The values in this view flow in the following direction: the value for the field `String authType` in `SingleSignonMessage` comes from `this.authType = authType` assignment statement in `SingleSignOnMessage.setAuthType()` method called from `ClusterSingleSignOn.register()` method with `authType` parameter and so on.
- Expand the tree to dig in the chain of assignments and method calls that lead to the symbol in question.
- Nodes with gray background denote duplicates (the usages that are already present in the tree in another location).

To find out what are the possible values a symbol can have, click Group By Leaf Expression button on the main tool bar of the Dataflow tool window.



Dataflow from Here

The hierarchical view is similar to the results of dataflow to here analysis, but the values flow in the opposite direction.


Analyze Dataflow from here expression null
(236, 17) return write(null); in BeanProperty.getWriteMethod()
(243, 34) private Method write(ELContext ctx) { in BeanProperty.write(ELContext)
(247, 50) throw new PropertyNotFoundException(message(ctx, in BeanProperty.write(ELContext
(31, 34) static String message(ELContext context, String name, Object[] props) { in ELResol

Validating Dependencies

Analyzing dependencies you might need to exclude some of them from the analysis process. In other words, you can create your own rules for dependencies analysis.

To validate dependencies

1. In the toolbar of the Dependency Viewer , click the Edit Rules button. Dependency Validation dialog is opened.

2. Define the scopes of prohibited and allowed usages:
   1. Click Add button in the Deny/Allow sections to add the default scope.
   2. Click a scope entry in the Deny/Allow usages of column and select one of the pre-defined scopes from the drop-down list, or click the ellipsis button and define scope in the Scopes dialog.
   3. Click a scope entry in the in / only in column, and select one of the pre-defined scopes, or click the ellipsis button and define scope in the Scopes dialog.
   4. Use Add and Remove buttons to make up the complete list.

3. Apply changes. Dependency analysis reruns with the new rules.

Code Inspection

In this section:

- Code Inspection
  - Code analysis basics
  - Inspection profiles
  - Inspection severity
  - Inspection scope
  - Examples of code inspections
    - Finding probable bugs
    - Locating dead code
    - Highlighting unused declarations
    - Unresolved JavaScript function or method
    - Examples of PHP Code Inspections
- Accessing Inspection Settings
- Customizing Profiles
- Changing the Order of Scopes
- Configuring Inspection Severities
- Running Inspections
- Running Inspection by Name
- Analyzing Inspection Results
- Applying Quickfixes Automatically
- Resolving Problems
- Suppressing Inspections
- Disabling and Enabling Inspections
- Exporting Inspection Results
- Running Inspections Offline
- Viewing Offline Inspections Results
- Changing Highlighting Level for the Current File
- Creating Custom Inspections

Code analysis basics

IntelliJ IDEA features robust, fast, and flexible static code analysis. It detects compiler and runtime errors, suggests corrections and improvements before you even compile.

IntelliJ IDEA performs code analysis by applying inspections to your code. Numerous code inspections exist for Java and for the other supported languages.

The inspections detect not only compiling errors, but also different code inefficiencies. Whenever you have some unreachable code, unused code, non-localized string, unresolved method, memory leaks or even spelling problems – you'll find it very quickly.

IntelliJ IDEA's code analysis is flexibly configurable. You can enable/disable each code inspection and change its severity , create profiles with custom sets of inspections, apply inspections differently in different scopes , suppress inspections in specific pieces of code, and more.

The analysis can be performed in several ways:

- By default, IntelliJ IDEA analyses all open files and highlights all detected code issues right in the editor. On the right side of the editor you can see the analysis status of the whole file (the icon in the top-right corner).
  When an error is detected, this icon is ; in case of a warning, it is ; if everything is correct, the icon is .
- Alternatively, you can run code analysis in a bulk mode for the specified scope, which can be as large as the whole project.
- If necessary, you can apply a single code inspection in a specific scope.

For the majority of the detected code issues, IntelliJ IDEA provides quick fix suggestions . You can quickly review errors in a file by navigating from one highlighted line to another by pressing `F2` `Shift+F2` .

For more information and procedural descriptions, see Configuring Inspection Severities .

Inspection profiles

When you inspect your code, you can tell IntelliJ IDEA which types of problems you would like to search for and get reports about. Such configurations can be preserved as inspection profiles.

An inspection profile defines the types of problems to be sought for, i.e. which code inspections are enabled/disabled and the severity of these inspections. Profiles are configurable in the Inspections settings page.

To set the current inspection profile (the one that is used for the on-the-fly code analysis in the editor), simply select it in the Inspections settings page and apply changes. When you perform code analysis or execute a single inspection , you can specify which profile to use for each run.

Inspection profiles can be applicable for the entire IDE or for a specific project:

- Project profiles are shared and accessible for the team members via VCS. They are stored in the project directory: `<project>/.idea/inspectionProfiles` .
- IDE profiles are intended for personal use only and are stored locally in XML files under the `USER_HOME/.<IntelliJ IDEA version>/config/inspection` directory.

If necessary, you can change the applicability of the profile (current project of IDE) and, consequently, its location with the Share profile check box (selected means current project) in the Inspections settings page.

IntelliJ IDEA comes with the following pre-defined inspection profiles:

- Default: This local (IDE level) profile is intended for personal use, applies to all projects, and is stored locally in the `Default.xml` file under the `USER_HOME/.<IntelliJ`

– Default: This local (IDE level) profile is intended for personal use, applies to all projects, and is stored locally in the `default.xml` file under the `USER_HOME/.<IntelliJ IDEA version>/config/inspection` directory.
– Project Default: when a new project is created, the Project Default profile is copied from the settings of a template project . This profile is shared and applies to the current project.

After a project is created, any modifications to the project default profile will pass unnoticed to any other projects.

When the settings of the Project Default profile are modified in the Template Project settings , the changed profile will apply to all newly created projects, but the existing projects will not be affected as they already have a copy of this profile.

Project Default profile is stored in the `Project_Default.xml` file located in the `<project>/.idea/inspectionProfiles` directory.

One can have as many inspection profiles as required. There are two ways of creating new profiles: you can add a new profile as a copy of the Project Default profile or copy the currently selected profile. The newly created profiles are stored in XML files, located depending on the type of the base profile .

The `<profile_name>.xml` files representing inspection profiles appear whenever some changes to the profiles are done and applied. The files only store differences against the default profile.

In case of the file-based project format , the shared profiles are stored in the project file `<project name>.ipr` .

Refer to the section Customizing Profiles for details.

Inspection severity

Inspection severity indicates how seriously the code issues detected by the inspection impact the project and determines how the detected issues should be highlighted in the editor. By default, each inspection has one of the following severity levels:

– Server problem
– Typo
– Info
– Weak Warning
– Warning
– Error

You can increase or decrease the severity level of each inspection. That is, you can force IntelliJ IDEA to display some warnings as errors or weak warnings. In a similar way, what is initially considered a weak warning can be displayed as a warning or error, or just as info.

You can also configure the color and font style used to highlight each severity level. Besides, you can create custom severity levels and set them for specific inspections.

If necessary, you can set different severity levels for the same inspection in different scopes .

All modifications to inspections mentioned above are saved in the inspection profile currently selected in the inspection settings and apply when this profile is used.

Inspection scope

By default, all enabled code inspections apply to all project files. If necessary, you can configure each code inspection (enable/disable , change its severity level and options) individually for different scopes . Such configurations, like any other inspection settings, are saved and applied as part of a specific profile .

There may be complicated cases when an inspection has different configurations associated with different scopes. When such inspection is executed in a file belonging to some or all of these scopes, the settings of the highest priority scope-specific configuration are applied. The priorities are defined by the relative position of the inspection's scope-specific configuration in inspection settings : the uppermost configuration has the highest priority. The Everywhere else configuration always has the lowest priority.

For more information and procedural descriptions, see Configuring Inspection for Different Scopes .

Examples of code inspections

In the Inspections page, all inspections are grouped into categories. The most common tasks covered by code analysis are:

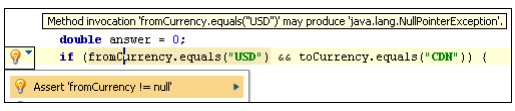
– Finding probable bugs.
– Locating dead code.
– Detecting performance issues.
– Improving code structure and maintainability.
– Conforming to coding guidelines and standards.
– Conforming to specifications.

Finding probable bugs

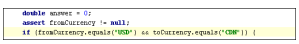
IntelliJ IDEA analyzes the code you are typing and is capable of finding probable bugs as non compilation errors right on-the-fly. Below are the examples of such situations.

For example, potential NPE can be thrown at runtime:

Before After



Here the first if condition may lead to a `NullPointer` exception being thrown in the second if, as not all situations are covered. At this point adding an assertion in order to avoid a NullPointer being thrown during the application runtime would be a good idea.

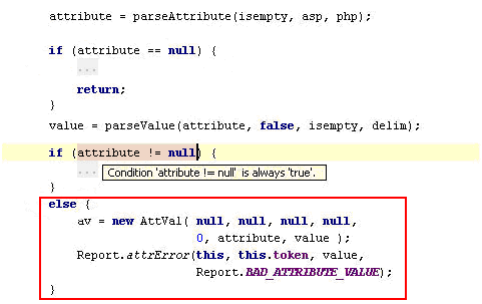


So, this is exactly what we get from the intention action.

Locating dead code

IntelliJ IDEA highlights in the editor pieces of so-called *dead* code. This is the code that is never executed during the application runtime. Perhaps, you don't even need this part of code in your project. Depending on situation, such code may be treated as a bug or as a redundancy. Anyway it decreases the application performance and complicates the maintenance process. Here is an example.

So-called *constant conditions* - conditions that are never met or are always *true*, for example. In this case the responsible code is not reachable and actually is a *dead* code.



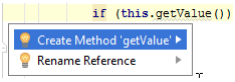
IntelliJ IDEA highlights the if condition as it's always true. So the part of code surrounded with else is actually a dead code because it is never executed.

Highlighting unused declarations

IntelliJ IDEA is also able to instantly highlight Java classes, methods and fields which are unused across the entire project via Unused declarations inspection. All sorts of Java EE `@Inject` annotations, test code entry points and other implicit dependencies configured in the Unused declarations inspection are deeply respected.

Unresolved JavaScript function or method

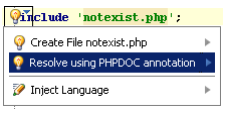
This inspection detects references to undefined JavaScript functions or methods.



Examples of PHP Code Inspections

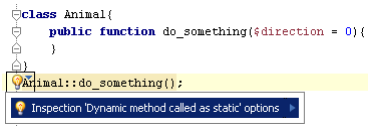
- Unresolved Include

  This inspection detects attempts to include not actually existing files and suggests two quick fixes: to create a file with the specified name or use a PHPDOC annotation.



- Dynamic method is called as static

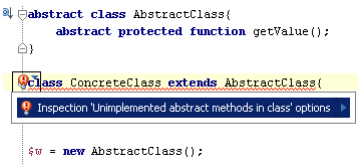
  This inspection checks whether a call to a static function is actually applied to a static function.



  The function `do_something()` is called as static while actually it is dynamic.

- Unimplemented abstract method in class

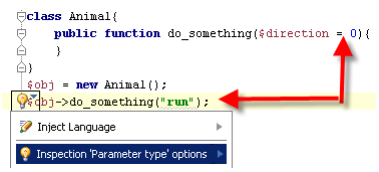
  This inspection checks whether classes inherited from abstract superclasses are either explicitly declared as abstract or the functions inherited from the superclass are implemented.



  The class `ConcreteClass` is inherited from an abstract class `AbstractClass` and has not been explicitly declared as abstract. At the same time, the function `GetValue()`, which is inherited from `AbstractClass`, has not been implemented.

- Parameter type

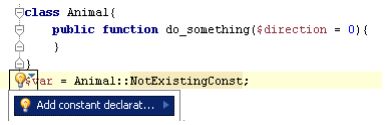
  PHP variables do not have types, therefore basically parameter types are not specified in definitions of functions. However, if the type of a parameter is defined explicitly, the function should be called with parameters of the appropriate type.



  The function `do_something` has the parameter of the type `integer` but is called with a `string`.

- Undefined class constant

  This inspection detects references to constants that are not actually defined in the specified class.



The constant `NotExistingConst` is referenced to as a constant of the class `Animal`, while actually it is not defined in this class.

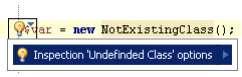
– Undefined constant inspection

This inspection detects references to constants that are not actually defined anywhere within the inspection scope.

The referenced constant `UndefinedConst` is not defined anywhere within the inspection scope.

– Undefined class

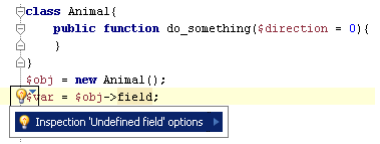
This inspection detects references to classes that are not actually defined anywhere within the inspection scope.



The referenced class `NotExistingClass` is not defined.

– Undefined field

This inspection detects references to fields of a class that are not actually defined in this class.



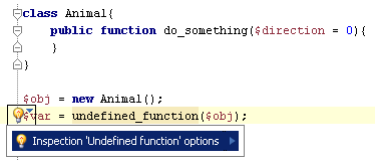
The `$obj` variable is an instance of the class `Animal`. The declaration of the `$var` contains a reference to the `field` field of the class `Animal`, which is not defined on this class.

To suppress reporting Undefined method errors in such cases, re-configure the inspection severity. To do that, open the Inspections page of the Settings dialog box, click the inspection name in the list and select the Downgrade severity if __magic methods are present in class check box in the Options area. After that undefined properties in such cases will be indicated one step lower than specified for inspections in general, by default, Info instead of Warning.

To suppress irrelevant reporting of Undefined field errors, clear the Notify about access to a field via magic method and Notify about PHP dynamic field declaration check boxes. When the check boxes are selected, IntelliJ IDEA reports errors even if the class contains the `__get()` and `__set()` magic methods.

– Call of undefined function

This inspection detects references to functions that are not defined anywhere within the inspection scope.



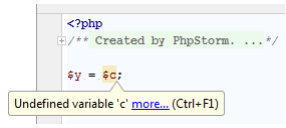
The called function `undefined_function()` is not defined anywhere within the inspection scope.

– Undefined variable

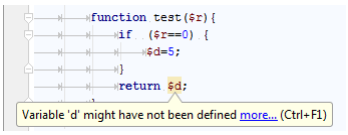
This inspection detects references to variables that are not declared and initialized anywhere within the inspection scope. PHP does not require that each variable should be declared and initialized. PHP can initialize such variable on the fly and assign it the zero value. However, this inspection allows you to detect discrepancies of this kind.

The Undefined variable inspection can be configured through the check boxes on the Inspections page of the Settings dialog:

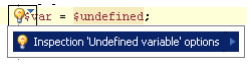
– Enable inspection in global space: select this check box to run the inspection against variables outside functions/methods, classes, and namespaces, that is, in the global space.



– Report that variable might have not been defined: select this check box to have a warning displayed even if the definition of a variable is not definitely missing. Such situation may take place when a variable is used in several paths and some of them may never be reached, for example, in `if()` statements:



– Ignore 'include' and 'require' statements Suppose the inspection scope contains an `include` or `require` statement. If this check box is cleared, IntelliJ IDEA treats such variable as defined in the classes referenced through these statements and no error is reported. If this check box is selected, an Undefined variable error is reported.

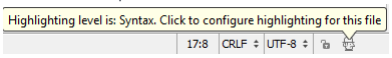


Accessing Inspection Settings

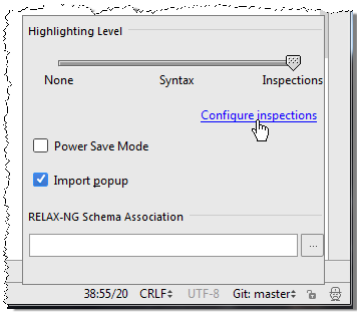
Inspections and inspection profiles are editable in the Inspections settings page. IntelliJ IDEA provides several ways to gain access to inspection settings.

To access inspections and profiles settings, do one of the following

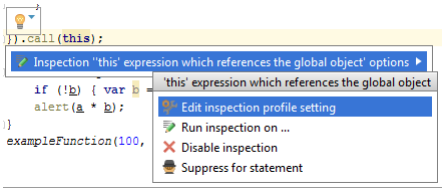
- Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Inspections under Editor.
- On the main toolbar, click , then expand the Editor node, and click Inspections .
- Click the current profile icon in the Status bar



and then click the Configure inspection link.



- In the editor, open the suggestion list, click the right arrow, and choose Edit inspection profile settings on the submenu.



- In in the Inspection tool window , click Edit Settings on the toolbar or use the corresponding context menu command.

Customizing Profiles

On this page:

– Introduction
– Customizing profiles
– Creating profiles
– Copying profiles
– Deleting profiles
– Importing and exporting profiles

Introduction

An inspection profile defines the types of problems to be sought for, i.e. which code inspections are enabled/disabled and the severity of these inspections. Profiles are configurable in the Inspections settings page.

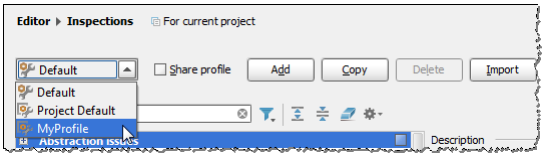
To set the current inspection profile (the one that is used for the on-the-fly code analysis in the editor), simply select it in the Inspections settings page and apply changes. When you perform code analysis or execute a single inspection , you can specify which profile to use for each run.

IntelliJ IDEA allows you to customize the existing inspection profiles (including default profiles), and create new ones. You can also share, import and export inspection profiles.

Customizing profiles

To customize a profile

1. In the inspection settings , select the profile to be changed.



   Note that the default profiles are also editable.

2. If necessary, you can change the applicability of the profile (current project of IDE) and, consequently, its location with the Share profile check box (selected means current project) .
3. Customize the desired inspections: enable or disable it , change its severity and the other options, which can be different for the various inspections.
   Note that the selectors of severity and scopes, and the options section (if any) are only available for the enabled inspections.
4. To apply an inspection to a restricted set of files, associate it with the corresponding scopes .
   By default, all inspections apply to all the sources of the current project.
5. Apply changes.

Creating profiles

To create a new profile

1. In the inspection settings , click Add.
2. In the dialog box that opens, type the name of the new profile.
3. Apply changes. The new profile is created as the copy of the Default profile.

Copying profiles

To create a copy of an existing profile

1. In the inspection settings , select the profile that you want to copy.
2. Click Copy.
3. In the dialog box that opens, type the name of the new profile.
4. Apply changes. The new profile is created as the copy of the currently selected profile.

Deleting profiles

To delete a profile

1. In the inspection settings , select the profile to be deleted from the list of profiles.
2. Click Delete.
3. Apply changes.

Importing and exporting profiles

To import a profile

1. In the inspection settings , click Import.
2. In the Select Path dialog box, locate the desired `xml` file, and click OK.
3. Apply changes.

To export a profile

1. In the inspection settings , select the profile that you want to export.
2. Click Export.
3. In the Select Path dialog box, locate the desired directory, where the `xml` file of the current inspection profile will be saved, and click OK.
4. Apply changes.

Configuring Inspection for Different Scopes

On this page:

– Basics

– Defining the order of scopes

Basics

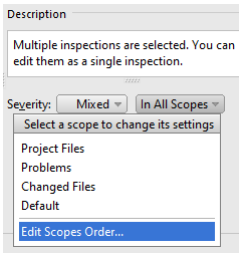
By default, all enabled code inspections apply to all project files. If necessary, you can configure each code inspection (enable/disable , change its severity level and options) individually for different scopes . Such configurations, like any other inspection settings, are saved and applied as part of a specific profile .

There may be complicated cases when an inspection has different configurations associated with different scopes. When such inspection is executed in a file belonging to some or all of these scopes, the settings of the highest priority scope-specific configuration are applied. The priorities are defined by the relative position of the inspection's scope-specific configuration in inspection settings : the uppermost configuration has the highest priority. The Everywhere else configuration always has the lowest priority.

Defining the order of scopes

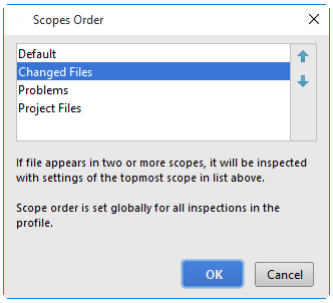
To define the order of scopes, follow these steps

1. In the Inspections page of the Settings/Preferences dialog, click the button In All Scopes:



2. Choose Edit Scopes Order... from the scopes drop-down list.

3. In the Scopes Order dialog box that opens, select the desired scope, and click the up and down arrows ⬆⬇:



Configuring Inspection Severities

On this page:

– Basics
– Changing severity of an inspection
– Changing severity of an inspection for different scopes
– Changing the highlighting style for a specific severity level
– Defining a custom severity level

Basics

Inspection severity indicates how seriously the code issues detected by the inspection impact the project and determines how the detected issues should be highlighted in the editor. By default, each inspection has one of the following severity levels:

– Server problem
– Typo
– Info
– Weak Warning
– Warning
– Error

You can increase or decrease the severity level of each inspection. That is, you can force IntelliJ IDEA to display some warnings as errors or weak warnings. In a similar way, what is initially considered a weak warning can be displayed as a warning or error, or just as info.

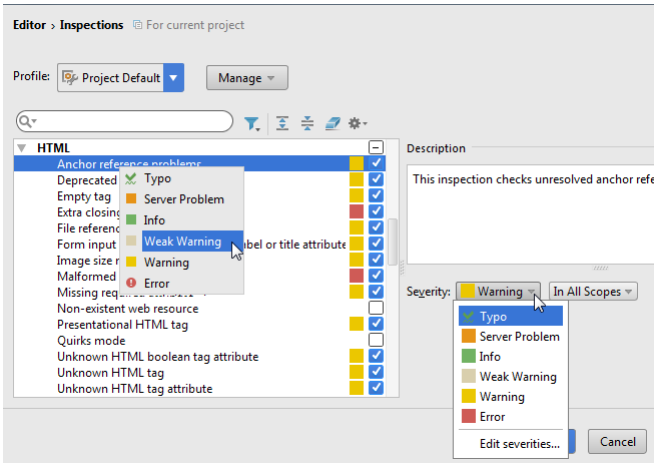
You can also configure the color and font style used to highlight each severity level. Besides, you can create custom severity levels and set them for specific inspections.

If necessary, you can set different severity levels for the same inspection in different scopes .

All modifications to inspections mentioned above are saved in the inspection profile currently selected in the inspection settings and apply when this profile is used.

Changing severity of an inspection

1. In the inspection settings , select the desired profile . The inspections associated with the profile are displayed in the tree view.
2. Select the desired inspection. If this inspection is disabled, select the check box next to it.
3. Select the desired severity from the context menu of the inspection or from the Severity selector on the right:

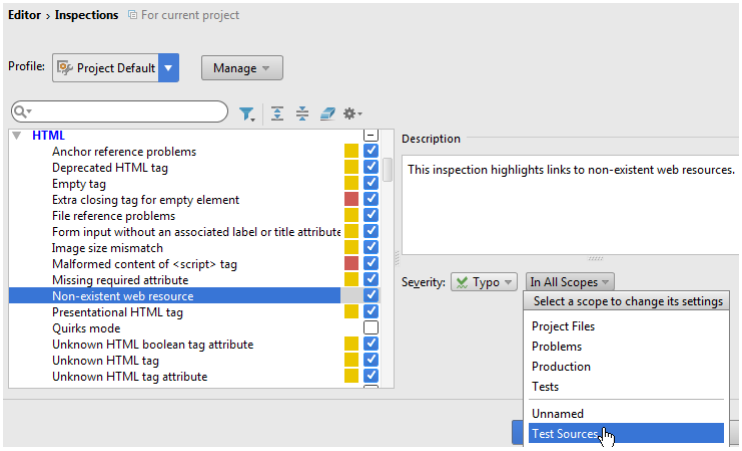


Note that inspections, whose state is changed relative to the default values, and all their grouping nodes are highlighted with blue.

4. Apply the changes. The modified inspection will now have the new severity level when this profile is used.

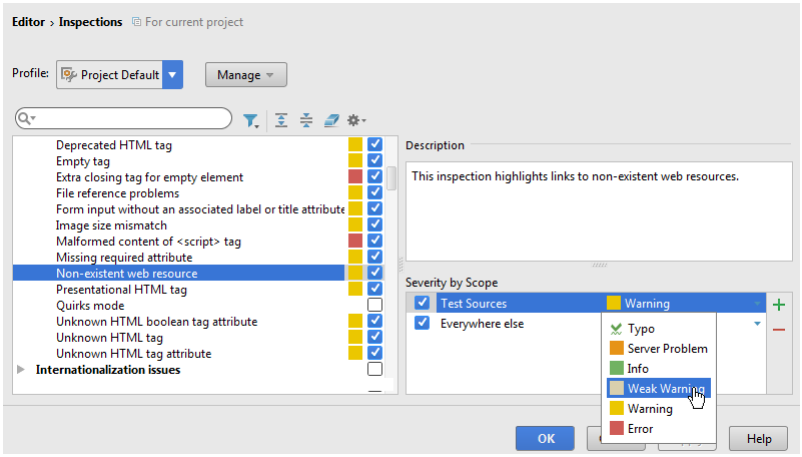
Changing severity of an inspection for different scopes

1. Choose the desired profile and inspection.
2. Click the drop-down list In All Scopes, and select the scope you want to change inspection severity in:



IntelliJ IDEA shows severities for two scopes: for the selected one and Everywhere else

3. Click severity drop-down list for the selected scope and choose the appropriate severity level from the drop-down list:



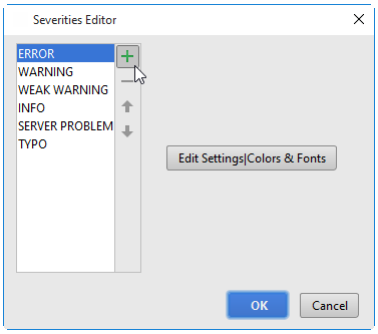
Changing the highlighting style for a specific severity level

1. Do one of the following:
   - In the Settings/Preferences dialog, select Editor | Colors & Fonts -> General, and then select the style corresponding to the desired severity level.
   - In the inspection settings , select the desired inspection and choose Edit severities from the Severity selector.
     Next, in the Severities Editor dialog box that opens, select the desired severity level and click Editor | Colors & Fonts.

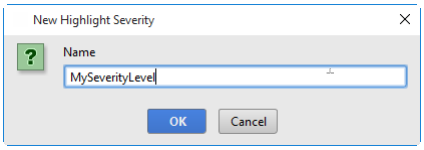
   Either way you will see the styles associated with this severity in the Colors and Fonts settings page.
2. Configure the color and font styles as necessary and apply changes. The detected issues with the corresponding severity will now be highlighted in the editor with the modified style when the current profile is used.

Defining a custom severity level

1. In the inspection settings , select the desired inspection and choose Edit severities from the Severity selector. .
2. In the Severities Editor dialog box that opens, click +:



3. Type the name for the new severity in the New Highlight Severity dialog box.



   The custom severity is added to the list of severities.
4. Specify color and font settings for the new severity using the controls to the right of the list of severities.
5. Use the Up and Down buttons to change the priority of the new severity.
6. Apply the changes. The new severity level will now be available for all inspections within the current profile . You can assign it to specific inspections and get the corresponding code issues highlighted with the specified style in the editor.
   If necessary, you can remove the custom severity level later by selecting it in the Severities Editor dialog box and clicking –.

   You cannot change priorities of the pre-defined severity levels, or remove them.

Running Inspections

Although code analysis is performed on-the-fly in all open files, you may want to run it for the whole project or a custom scope and examine the results in a friendly view.

Note Inspecting code in a large scope, for example, the whole project, can be time consuming. For large projects, it is recommended to create a number of smaller scopes and run analysis in each of them separately.

– If you do not want to have some code issues in the analysis report, you can disable some inspections .

To run a code inspection

1. Open the desired file in the editor. Alternatively select files or directories in the Project tool window . For multiple selection, click the items holding down the Ctrl / ⌘ key. The initial inspection scope will be confined to the opened file or the selection.

2. On the main menu, choose Analyze | Inspect Code . The Specify Inspection Scope dialog box opens.

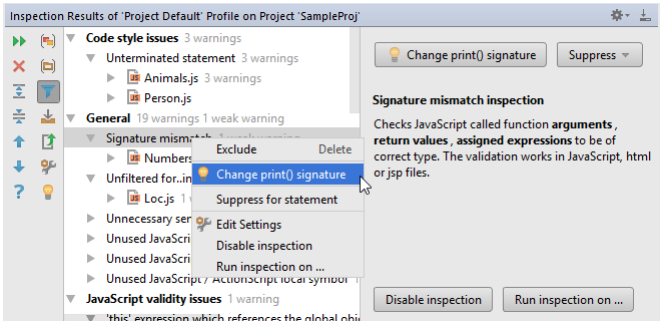
3. In the Inspection scope area, specify which files should be inspected.
   – To have the source code of the entire project inspected, select the Whole Project option.
   – If you are using version control integration , you can choose to only inspect uncommitted files.
   – To run an inspection for the currently opened file, or the file(s)/folder(s) selected in the Project view, select the File/Module <name> option.
   – To apply inspect code in a specific scope , select the Custom scope option, then choose the desired scope from the drop-down list or click the Browse button ... and configure a new scope in the Scopes dialog box.

4. To have test source files inspected too, select the Include test sources check box.

5. Specify the inspection profile to apply. Do one of the following:
   – Select one of the existing profiles from the Inspection Profile drop-down list.
   – Click the Browse button ... and configure a new profile in the Inspections dialog box.

6. Click OK to run code analysis.

7. Examine results in the Inspection tool window.



It is also possible to run inspections offline, without starting the IDE. Follow the procedure described in the section Working with IntelliJ IDEA Features from Command Line .

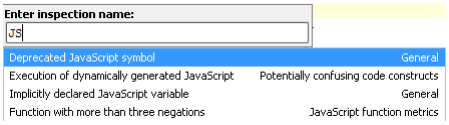
Running Inspection by Name

Rather than running all enabled inspections, IntelliJ IDEA makes it possible to exactly specify the desired inspection by its name and run it to inspect the code in the specific scope.

To run a code inspection by name

1. On the main menu, choose Analyze | Run Inspection by Name , or press `Ctrl+Shift+Alt+I` .
2. In the pop-up frame that opens, start typing the inspection name. As you type, the suggestion list shrinks to show the matching inspection only.

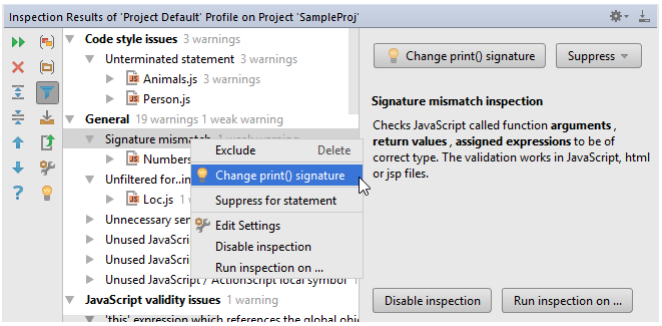
   Tip Use CamelHumps to match camel case words and white spaces to match initial letters of the words.

   The Specify Inspection Scope dialog box opens.
3. In the Inspection scope area, specify which files should be inspected.
   - To have the source code of the entire project inspected, select the Whole Project option.
   - If you are using version control integration , you can choose to only inspect uncommitted files.
   - To run an inspection for the currently opened file, or the file(s)/folder(s) selected in the Project view, select the File/Module <name> option.
   - To apply inspect code in a specific scope , select the Custom scope option, then choose the desired scope from the drop-down list or click the Browse button ... and configure a new scope in the Scopes dialog box.
4. To have test source files inspected too, select the Include test sources check box.
5. To apply inspection only in files matching the specific mask, select the File Masks(s) and specify the file mask. Use comma to separate multiple file masks.
6. Click OK to run the inspection.
7. Examine results in the Inspection tool window.

Analyzing Inspection Results

After you perform code analysis or execute a single inspection , you can examine the results in the Inspection tool window .



Each run of the code analysis or a single inspection is displayed in a new tab in the tool window. The left part of the tab displays detected code issues groped by inspection groups, inspections and files. If necessary, you can change the default grouping. When you select a specific issue, its report appears in the right part.

In the Inspection tool window you can:

- Resolve issues using quick-fixes where available.
- Jump to the source of the problem to resolve it manually.
- Export inspection results to an XML or HTML file.
- Disable inspections to skip all corresponding issues nex time you ran code analysis.
- Suppress inspections for specific code issues.
- Access inspection settings to configure inspections.

Applying Quickfixes Automatically

On this page:

– Introduction
– Applying quickfixes automatically
– Applying quickfixes on commit to VCS

Introduction

When you run an inspection against your code, the Inspection Tool Window is displayed showing the results of code analysis. You can then examine the issues that have been detected and apply the suggested quickfixes .

However, you may want to streamline this process by automatically applying the quickfixes from your inspection profile to the selected scope, without having to examine the results and implement each separate fix individually.

Applying quickfixes automatically

To apply quickfixes automatically

1. From the main menu, select Analyze | Code Cleanup .
2. In the Specify Code Cleanup Scope dialog that opens, select ths scope to which you want the inspection profile to be applied.
3. Select the inspection profile from the drop-down list, or click the ... button to configure a new profile in the Inspections dialog that opens. You can also click the ... button to check which fixes will be applied.
4. Click OK to launch the inspection.

IntelliJ IDEA will perform code analysis and will automatically apply quickfixes from the selected inspection profile to all issues that are detected.

Alternatively, you can place the caret at an error in the source code that corresponds to a quickfix, click the red bulb (quick-fix suggested) icon that appears on the left, and select Code Cleanup from the drop-down menu. Code cleanup will be performed for the current file with the current inspection profile.

Applying quickfixes on commit to VCS

To apply quickfixes when committing changes to a VCS

1. From the main menu, select VCS | Commit Changes.
2. In the Commit Changes dialog that opens, select the files that you want to commit, and select the Cleanup code option in the Before Submit area.

Code cleanup will be performed for all selected files before they are committed to your version control system (the current project inspection profile will be applied).

Resolving Problems

On this page:

- Introduction
- Fixing problems

Introduction

By default, IntelliJ IDEA analyses the code in all open files and highlights the detected code issues. You can fix most of the issues immediately by applying quick-fixes .

If you perform code analysis or execute a single inspection in a larger scope of source files, IntelliJ IDEA displays the detected code issues in the Inspection tool window. When you select a specific issue in this window, its report is shown in the right part of the window.

If there are available fixes to the issue, IntelliJ IDEA notifies it in the following ways:

- The Apply a Quickfix button becomes available on the Inspection's window toolbar.
- The available fixes are shown in the optional Problem resolution field of the report.
- The available fixes are shown in the context menu of the issue.

If there are no available fixes, the only option is to fix the issue manually.

Please note the following:

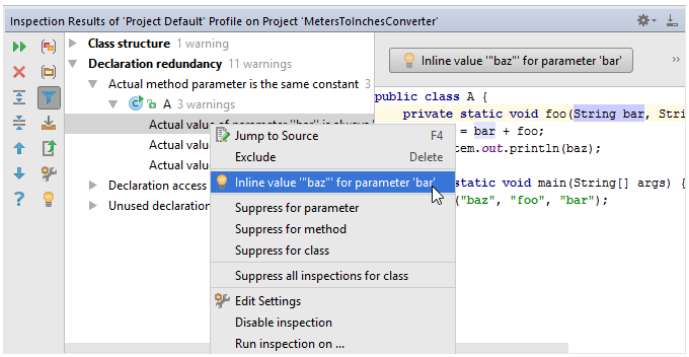
- To display the source code of an issue in the editor, when it is selected in the Inspections tool window, toggle the Autoscroll to Source button.
- If you find that some of the reported issues are minor or not helpful to you, you can ignore them either by disabling the corresponding inspection or by suppressing it in a specific piece of code.

Fixing problems

To fix a problem reported by code inspection

In the Inspections tool window, select the code issue you are interested in and do one of the following:

- If IntelliJ IDEA suggests any fixes to the issue as described above, you can use one of them to fix the problem immediately.



- If no resolutions are suggested, use the Jump to source command in the context menu and fix the problem manually.

Suppressing Inspections

On this page:

– Introduction

– Suppressing inspections in the editor

– Suppressing inspections from the Inspection tool window

Introduction

For some reasons, you may want to partly disable a specific inspection, i.e. ignore some code issues while continuing to detect the other issues with this inspection.

For example, IntelliJ IDEA considers some code to be "dead", and you can see that it is true. The inspection is helpful and you do not want to disable it. However, you may want to use this code later and do not want it to be highlighted in the editor or appear in the issue reports.

To do so, IntelliJ IDEA allows you to suppress certain inspections for a specific statement, function/method, tag or file. You can do it either in the editor , using the suggestion list or in the Inspection tool window when analysing inspection results.

Let's summarize the difference between suppressing and disabling code inspections:

When suppressing an inspection, IntelliJ IDEA inserts a special comment that tells the code analysis engine to ignore the issues found by this inspection in the specific piece of code.

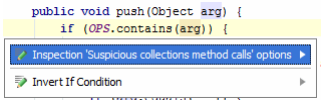
When disabling an inspection, you just turn it off so the code analysis engine just ignores the code issues found by this inspection.

Some code inspections (e.g. those detecting errors) cannot be suppressed.

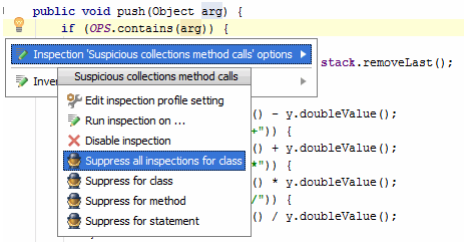
Suppressing inspections in the editor

To suppress an inspection in the editor

1. Set the cursor to the highlighted code issue in the editor.
2. Press `Alt+Enter`, or click the light bulb icon to expand the suggestion list.



3. Depending on the issue, you will see either quick-fixes related to the inspection or the Inspection "<inspection name>" options item.
4. Use the up/down arrow keys to select this item and then press the right arrow key or just click the right arrow ▸ next to this item.
   Pressing the left arrow key, or `Escape` hides the suggestion list.
5. In the inspection options list, select the desired suppress action:

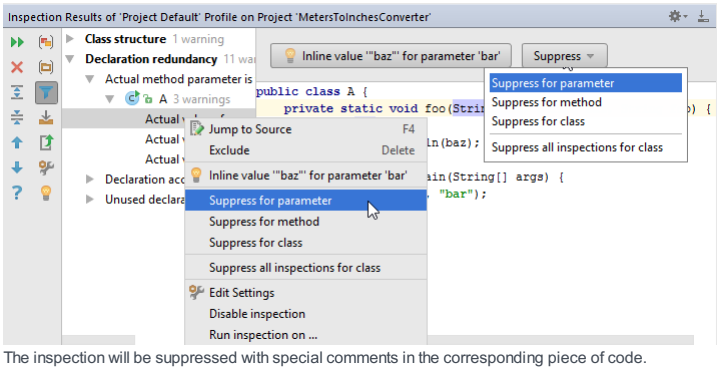


The inspection will be suppressed with special comments in the corresponding piece of code.

Suppressing inspections from the Inspection tool window

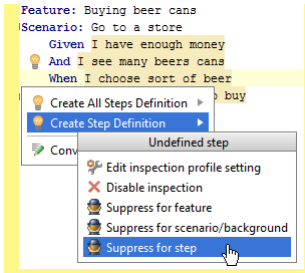
To suppress inspections from the Inspection tool window

1. After running code analysis , select a code issue, for which you want to suppress the inspection, in the Inspection tool window.
2. Click the button Suppress and choose the scope of sustension, or just right-click the selected inspection.
3. Choose the desired suppress action. For example:



The inspection will be suppressed with special comments in the corresponding piece of code.

Note The set of suppress actions depends on the language to which inspection applies. For example, for Cucumber, one can suppress inspections for a whole feature, for a scenario, or for a particular step:


Feature: Buying beer cans
Scenario: Go to a store
Given I have enough money
And I see many beers cans
When I choose sort of beer
buy
Create All Steps Definition
Create Step Definition
Undefined step
Edit inspection profile setting
Disable inspection
Suppress for feature
Suppress for scenario/background
Suppress for step

Disabling and Enabling Inspections

On this page:

– Introduction

– Disabling or enabling inspections

Introduction

If you think that some inspections report about the problems that you are not interested in, you can disable such inspections. Note that when you disable an inspection, it is disabled in the current inspection profile ; in all other profiles, it remains enabled.

There are several ways to disable/enable inspections:

– Using the Inspections page in the Settings/Preferences dialog - this is the main interface for managing inspections; here you can see at once, which inspections are enabled or disabled in all inspection profiles.

– Using the intention actions - this is the way to disable a highlighted code issues right in the editor.

– In the Inspection tool window - this is a quick way to disable uninteresting issues when analyzing inspection results . Note that here you can only disable inspections.

Note Note the difference between disabling and suppressing code inspections:

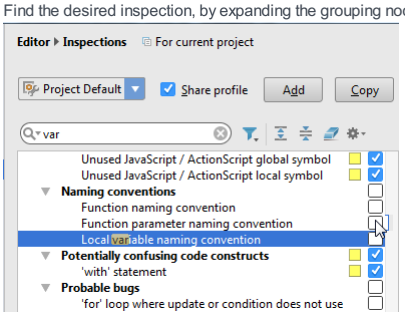
When suppressing an inspection, IntelliJ IDEA inserts a special comment that tells the code analysis engine to ignore the issues found by this inspection in the specific piece of code.

When disabling an inspection, you just turn it off so the code analysis engine just ignores the code issues found by this inspection.

Disabling or enabling inspections

To disable or enable an inspection in the Settings/Preferences dialog

1. Find the desired inspection, by expanding the grouping nodes or using the search field.



2. Use the check box next to the inspection to disable or enable it.

3. Apply the changes and close the dialog box.

To disable an inspection for highlighted issue in the editor

– When you disable inspections this way, they are disabled for the current inspection profile .

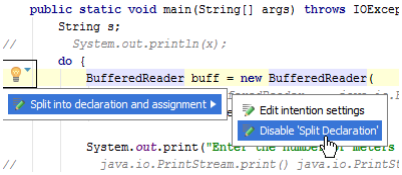
– To re-enable inspections disabled this way, use the main procedure described above.

1. Set the caret at a highlighted issue.

2. Click the bulb icon or press `Alt+Enter` to reveal the inspection alert and suggestion list.

3. Select the inspection to be disabled, then click right arrow button or just press the right arrow key.

4. On the submenu, click Disable <inspection name>.



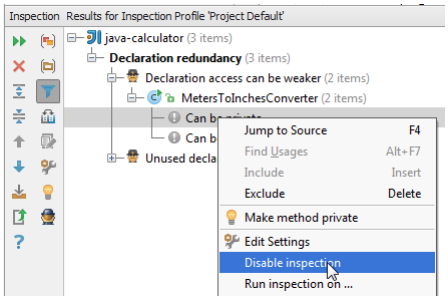
To disable inspections from the Inspection results report

– When you disable inspections this way, they are disabled for the inspection profile that was used for running inspections . You can see it in the header of the Inspection window's tab.

– To re-enable inspections disabled this way, use the main procedure described above.

1. In the Inspection tool window , right-click the inspection you want to disable.

2. On the context menu, choose Disable inspection.



3. Press the filter button to hide the disabled inspection alerts.

Exporting Inspection Results

After you perform code analysis or execute a single inspection , you can save the inspection results for further examination or for sharing with colleagues. IntelliJ IDEA enables you to export inspection results to the HTML or XML format.

To export inspection results

1. On the toolbar of the Inspection tool window, click the Export button .
2. From the Export To context menu, select the target format. The available options are HTML and XML.
3. In the dialog that opens , specify the target directory to store the inspection results in.

Running Inspections Offline

On this page:

- Basics
- Launching a code inspection from the command line
  - Examples
    - Windows
    - OS X
- Viewing the results of an offline inspection

Basics

In addition to running code inspections from the main menu, or from the context menus of the Project tool window , you can also launch the inspector from the command line, without actually running IntelliJ IDEA.

This way you can perform regular code inspections as a part of your development process, which is especially important for large projects. Inspection results are stored in the XML format.

Launching a code inspection from the command line

To launch a code inspection from the command line

- Specify the following command line arguments:
  - Path to the launcher: specify the full path to one of the following launchers (which reside under the `bin` directory of your IntelliJ IDEA installation):
    - For Windows: `inspect.bat`
    - For UNIX and OS X: `inspect.sh`
  - Project file path is the full path to the directory that contains the project to be inspected.
  - Inspection profile path is the full path to the profile, against which the project should be inspected. The inspection profiles are stored under `USER_HOME\.IntelliJ IDEAXX\config\inspection`
  - Output path is the full path to an existing directory where the report will be stored.
  - Options. You can specify:
    - The directory to be inspected `-d <full path to the subdirectory>`
    - The verbosity level of output `-vX` , where X is 0 for quiet, 1 for noisy and 2 for extra noisy.

Warning! Please note that you have to specify full paths. Relative paths are not accepted!

Tip If SDK is not defined, the inspection will fail. The SDK descriptions should be stored in `config\options\jdk.table.xml` . Learn how to configure SDK here .

Examples

Windows

```
C:\Program Files (x86)\JetBrains\<IntelliJ IDEA home>\bin\inspect.bat
E:\SampleProjects\MetersToInchesConverter E:\Work\MyProject\.idea\inspectionProfiles\Project_Default.xml
E:\Work\MyProject\inspection-results-dir -v2 -d E:\SampleProjects\MetersToInchesConverter\subdirectory
```

OS X

```
/Applications/IntelliJ IDEA.app/Contents/bin/inspect.sh ~/IntelliJ IDEAProjects/MyTestProject
~/Library/Preferences/ideaXX/inspection/Default.xml ~/IntelliJ IDEAProjects/MyTestProject/results-dir -v2
```

Viewing the results of an offline inspection

If you have performed an offline inspection and exported the inspection results to a directory in the XML format you can always download and view these results.

To view the results of an offline inspection, follow these steps

1. Open the project against which the inspection was performed.
2. On the main menu, choose Code | View Offline Inspection Results.
3. In the Select Path dialog box that opens, navigate to the directory that contains inspection results in XML format.
4. Click OK. Inspection results display in the Offline View tab in the Inspection tool window .

Tip Alternatively, you can open the relevant XML file in IntelliJ IDEA or in any other text processor without opening the inspected project.

Viewing Offline Inspections Results

If you have exported inspection results to a directory in XML format, or performed offline inspection outside IntelliJ IDEA, as a part of the automated build process, you can always download and view the results.

– Prerequisite

– Viewing offline inspection results

Prerequisite

The relevant project should be opened in the IDE, and should contain the classes that have been inspected. Otherwise, you have to review inspection results, opening the reports as XML files in the editor.

To view inspection results offline

1. On the main menu, choose Analyze | View Offline Inspection Results .
2. In the Select Path dialog, navigate to the directory that contains inspection results in XML format.
3. Click OK. Inspection results display in the Offline View tab in the Inspection tool window .

Changing Highlighting Level for the Current File

Use the Status bar to quickly re-configure highlighting for the file which is currently opened in the editor. With Hector, you can choose to highlight syntax problems, inspection issues, or turn off highlighting.

To change the highlighting level for the current file

1. Open the Highlighting level pop-up window by doing one of the following:
   - On the main menu, choose Analyze | Configure Current File Analysis .
   - Press `Ctrl+Shift+Alt+H`.
   - Click the Hector icon on the Status bar.
   - Right-click the code inspection indicator on top of the scroll bar.

2. Move the slider to one of the three available positions that define the highlighting level:
   - None: turn off problems highlighting in the editor.
   - Syntax: highlight syntax problems only.
   - Inspections: (default) highlight syntax problems and inspection issues.

Creating Custom Inspections

IntelliJ IDEA allows you to create custom inspections based on search templates . The inspection list provides a Structural Search Inspection, which is a powerful tool for creating search templates. Based on this inspection, you can create any number of search templates to inspect your source code and any number of replace templates to be used as means to resolve detected problems.

Tip Using the Structural Search Inspection is the only way to create custom inspections through the IntelliJ IDEA interface. Alternatively, you can develop an external plugin to implement a custom inspection.

To create a custom inspection

1. Open the Inspections dialog box .
2. Select the desired profile from the drop-down list. The inspections associated with the selected profile are displayed in the tree view.
3. In the list of inspections, expand the General node and select the check box next to the Structural Search Inspection item. This enables the controls in the Options area, where you can configure a custom inspection.
4. Click the Add Search Template button. In the Structural Search dialog box that opens specify the search options .
5. Click the Add Replace Template button. In the Structural Replace dialog box that opens specify the replace options.
6. Complete the list of inspection templates:
   - To edit a template, select it in the Selected Templates list, click the Edit button, and edit the template in the Edit Template dialog box that opens.
   - To remove a template, select it in the list and click the Remove button.
7. Specify the inspection severity using the Severity drop-down list.
8. Associate the inspection with a scope.
   If an inspection is not associated with a scope, it applies to all the sources of the current project.

Intention Actions

In this section:

- Intention Actions
  - Introduction
  - Intention action icons
  - Intention action types
- Applying Intention Actions
- Configuring Intention Actions
- Disabling Intention Actions

Introduction

IntelliJ IDEA helps you handle the situations when you use classes that haven't been imported, or methods that haven't been written etc., which can result in errors. When a possible problem is suspected, IntelliJ IDEA suggests a solution, and in certain cases can implement this solution (properly assign variables, create missing references and more). Besides syntax problems, IntelliJ IDEA recognizes code constructs that can be optimized or improved, and suggests appropriate intention actions, denoted with the special icons.

Intention action icons

Item	Icon	Description
Intention actions suggested		A yellow bulb indicates that IntelliJ IDEA just proposes to alter your code. It covers a range of situations from warning correction to suggestions for code improvement (like micro-refactorings).
Specific intention action		This sign appears in the suggestion list before each specific intention action. If an intention action alert is disabled, the sign turns to . Disabled intention action is still available and can be enabled again.
Quickfix suggested		A red bulb with an exclamation mark indicates that IntelliJ IDEA suggests a way to fix an error. It is related to Create from usage intentions and Quick fixes.
Disabled		Alert is disabled, but the intention action is still available and can be enabled again.

Intention action types

Find descriptions of specific intention actions on the Intentions page of the editor settings/preferences, where they are grouped according to the areas of their usage. Generally, intention actions can be divided into several categories, for example:

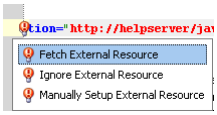
Create from usage

This type of intention action creates new code items: classes, methods, etc. They are smart enough to analyze your code and provide actions suitable for a particular case. The main concept behind this type is that you can begin using new things without declaring them first. You are not taken away from your current task for mundane minutiae like creating declarations, new files, etc. which IntelliJ IDEA handles while you keep focused.

For example, Create Constant Field is suggested if the reference is uppercase, or Create class appears when a name is typed after the `new` keyword, or when an identifier starts with a capitalized letter, etc.

Quick fixes

This type of intention action responds to common coding mistakes: using an improper access modifier, or an expression of the wrong type, or missing resources, etc. IntelliJ IDEA catches these kinds of problems as you type, and provides a quick way to fix them using Intentions Actions with appropriate suggestions for the error.



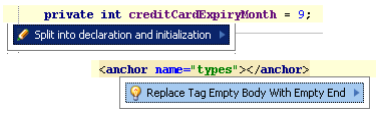
Micro-refactorings

These intention actions appear for code that is syntactically correct, but can be structurally improved by such things as:

- Converting code constructs.

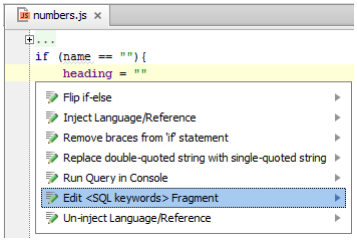
- Splitting declarations and assignments.

- Splitting or merging statements and tags, etc.



Edit <Injected Language> Fragment

For string literals that represent language injections , the Edit <Injected Language> Fragment intention action is available. You can use this intention action to open the corresponding code fragment in a separate editor.



Applying Intention Actions

IntelliJ IDEA displays an intention action alert, and it is your task to define which action (if any) should be performed.

To apply an intention action

1. Click the light bulb icon, or use the `Alt+Enter` keyboard shortcut to bring up the suggestion list.
2. Click the desired option, or use the arrow keys to select the option and press `Enter`.

Configuring Intention Actions

On this page:

– Configuring intention settings using the Settings/Preferences dialog
– Configuring intentions on-the-fly

Configuring intention settings using the Settings/Preferences dialog

To configure intention settings using the Settings dialog

IntelliJ IDEA makes it possible to configure intention action settings either in the Intentions page of the Settings dialog, or "on-the-fly".

By default, all available intention actions that ship with IntelliJ IDEA, are enabled. By changing the Intention Settings, you can disable the actions that are not required for your current working environment.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Intentions under Editor.
2. In the Intentions page, clear the check boxes of the intention actions or action categories that you do not currently need.
   Selecting or clearing a category affects all intention actions in this category.
3. Apply changes and close the dialog.

Configuring intentions on-the-fly

To configure intention settings on-the-fly

1. In the editor, press `Alt+Enter` to reveal the inspection alert and suggestion list.
2. Select the action to be disabled, then click right arrow button or just press the right arrow key.
3. On the submenu, click Disable <intention action name>.

Disabling Intention Actions

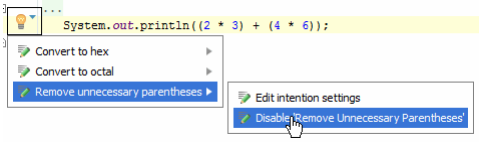
If an intention action is enabled, the alert shows automatically, when the caret rests on the problem-causing piece of code. You can disable alert for any type of intention actions, and show it by explicit invocation only.

Disabled intention actions are marked with the grey light bulb icon. Though disabled, such actions are still available and can be applied to the source code.

You can suppress intention actions related to inspections "on-the-fly", as described in the section Suppressing Inspections .

To disable an intention action alert

1. Click `Alt+Enter`, or click the light bulb icon to show the suggestion list.
2. Select the action to be disabled and click the right arrow button.
3. On the submenu, click Disable <intention action name>:



For the disabled intention action, the menu item changes to Enable <intention action name>.

Annotating Source Code

In this section you will find the basics of IntelliJ IDEA's annotations, and the related procedures:

- Enabling Annotations .
- @Nullable and @NotNull annotations, which are used, when IntelliJ IDEA supposes that a certain element can become `Null` .
- @NonNls annotation, which is used to ignore the hard-coded string literals.
- @Contract annotations, which is a powerful and flexible tool for making the APIs safer.
- Annotating Source Code Directly , which describes how to add annotations to your source code.
- Inferring Nullity , which describes how to analyze the source code for the possible `@Nullable` and `@NotNull` annotations.

Besides that, you can also find the following information:

- External Annotations that are stored outside the source code.
- Using External Annotations

Bundled IntelliLang plugin provides three new annotations you may find useful:

- `@Language` : responsible for the Language injection feature.
- `@Pattern` : is used to validate Strings against a certain regular expression pattern
- `@Subst` : is used to substitute non-compile time constant expressions with a fixed value.

To learn more about these IntelliLang annotations, refer to the plugin documentation .

Enabling Annotations

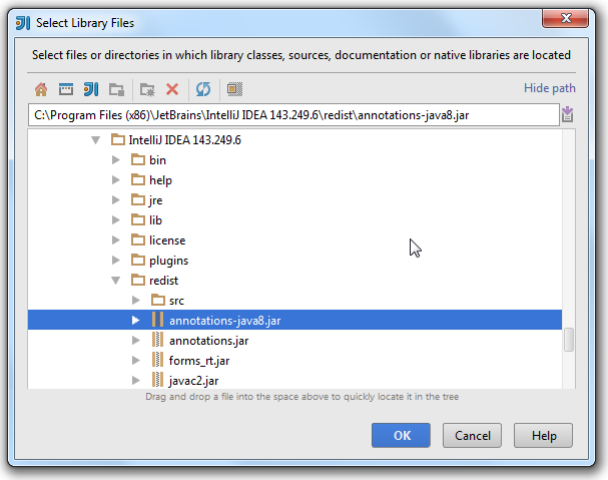
Introduction

IntelliJ IDEA lets you enable annotations in projects locally, through Maven repository for projects imported from Gradle or Maven model and through the editor. .

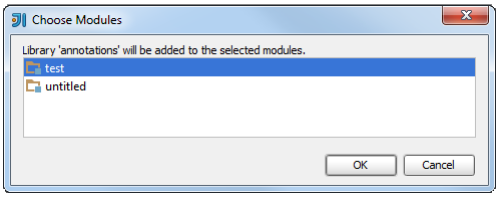
Enabling annotations in the project

To enable annotations, follow these steps:

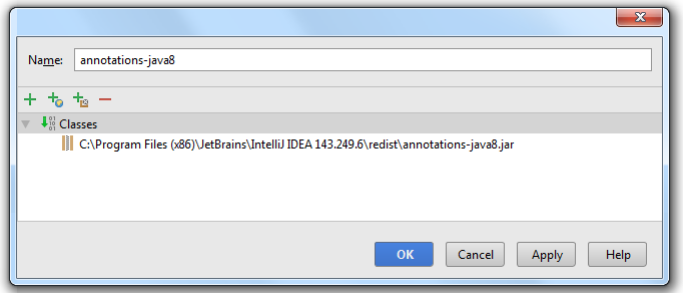
1. Click to open the project structure dialog .
2. In the Project Structure dialog box, click Libraries .
3. Under that <IntelliJ IDEA home> directory, select `redist/annotation .jar` , and click OK:



4. Click +. Select Library Files dialog box opens.
5. If your project contains more than one module, choose the desired module to be modified, and click OK:



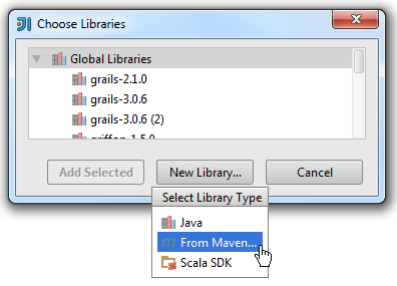
6. The new library is added to the classpath:



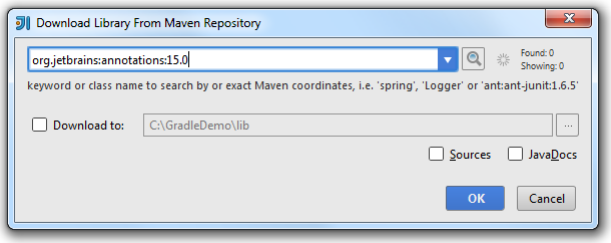
Enabling annotations in Gradle or Maven projects

To enable annotations in Gradle or Maven projects, follow these steps:

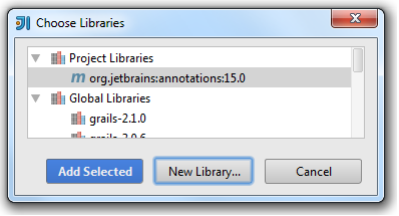
1. Click to open the project structure dialog.
2. In the Project Structure dialog box, on the Modules page, click the Dependencies tab.
3. On the Dependencies tab, click + and from the list that opens, select Library.
4. In the Choose Library dialog, click New Library and from the Select Library Type list, select From Maven.



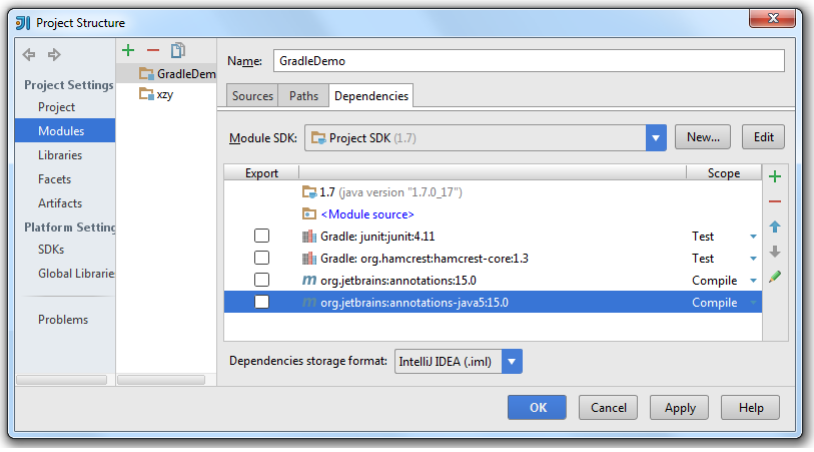
5. In the Download Library From Maven Repository dialog, enter `org.jetbrains:annotations:15.0` that contains annotations for Java 8 or `org.jetbrains:annotations-java5:15.0` that contains annotations compatible with Java 5 and click OK.



6. In the Choose Libraries dialog box, click Add Selected.



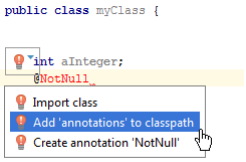
The new library is added to the list of dependencies and to the classpath.



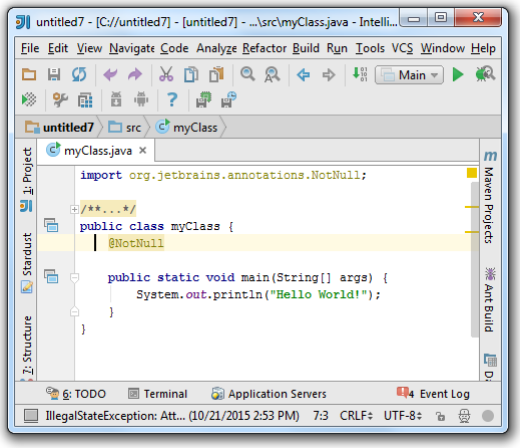
Enabling annotations using editor

To enable annotations in the editor:

– In the editor, write `@NotNull` and press Alt +Enter to invoke Add 'annotations' to classpath action.



IntelliJ IDEA will automatically choose Java 5 or Java 8 compatible JAR file based on your project's settings and add the JAR file to the project.



Note that if a project is imported from the Maven model, IntelliJ IDEA will add the dependency to the `pom.xml` .

@Nullable and @NotNull Annotations

Overview

This section describes @Nullable and @NotNull annotations introduced in IntelliJ IDEA for catching NullPointerException's (NPE's) through the Constant Conditions & Exceptions and @Nullable problem inspections.

These annotations are designed to help you watch contracts throughout method hierarchies to avoid emergence of NPE's. Moreover, IntelliJ IDEA provides another benefit for them: the Code Inspection mechanism informs you on such contracts' discrepancies in places where annotated methods are called and provides automated solutions in some cases.

Two annotations - @Nullable and @NotNull - handle method invocations and field dereferences outside methods.

@Nullable

The @Nullable Annotation reminds you about the necessity to introduce an NPE check when:

- Calling methods that can return null.
- Dereferencing variables (fields, local variables, parameters) that can be null.

@NotNull

The @NotNull Annotation is, actually, an explicit contract declaring the following:

- A method should not return null.
- A variable (like fields, local variables, and parameters) cannot hold null value.

IntelliJ IDEA warns you if these contracts are violated.

Tip For more information and code examples, refer to online how-to .

Formal Semantics

An element annotated with @Nullable claims null value is perfectly valid to return (for methods), pass to (for parameters) and hold (for local variables and fields).

An element annotated with @NotNull claims null value is forbidden to return (for methods), pass to (for parameters) and hold (for local variables and fields).

There is a covariance-contravariance relationship between @Nullable and @NotNull when overriding/implementing methods with annotated declaration or parameters.

- Overriding/implementing methods with an annotated declaration:
  - The @NotNull annotation of the parent method requires the @NotNull annotation for the child class method.
  - Methods with the @Nullable annotation in the parent method can have either @Nullable or @NotNull annotations in the child class method.
- Overriding/implementing methods with annotated parameters:
  - The @Nullable annotation of the parameter in the parent method requires the @Nullable annotation for the child class method parameter.
  - Methods with the @NotNull annotation of the parameter in the parent method can have either @Nullable or @NotNull annotations (or none of them) for the child class method parameter.

@NonNls Annotation

This section describes formal semantics and usage examples for @NonNls annotation introduced in IntelliJ IDEA. This annotation indicates that the annotated code element is not a string requiring internationalization and it does not contain such strings.

Formal Semantics

When an element is annotated with @NonNls, internationalization mechanisms will skip it or strings inside it.

annotated method parameter
: A string constant passed as a parameter in the method call is skipped. Also, if the property setter method parameter is annotated, such property values are not highlighted in the GUI Designer forms.

annotated field/variable
: String literals in field/variable initializer are skipped.

method is called on an annotated field/parameter/variable
: String literals passed as parameters to the method are skipped.

annotated field/parameter/variable passed as a parameter to the equals() method invoked on a string literal
: The string literal is skipped.

annotated field/parameter/variable at the left part of the assignment expression
: All string literals in the right part of the expression are skipped.

annotated method
: All string literals returned by the method are skipped.

annotated class
: All string literals in the class and all its subclasses are skipped.

annotated package
: All string literals in the package and all its subpackages are skipped.

@Contract Annotations

In this section:

– Syntax of the @Contract annotations

– Example

Syntax of the @Contract annotations

The `@Contract` annotation has two attributes: `value` and `pure`.

The `@Contract` annotation value has the following syntax:

```
contract ::= (clause  ;
                    )* clause
                    clause ::= args  ->  effect
                    args ::= ((arg  ,
                    )* arg )?
                    arg ::= value-constraint
                    value-constraint ::=  any
                    |  null  |  !null
                    |  false  |  true
                    effect ::= value-constraint
                    |  fail
```

The constraints here are:

```
_ - any value
                    null - null value
                    !null - a value
                    statically proved to be not-null
                    true - true boolean value
                    false - false boolean value
                    fail - the method throws
                    exception, if the arguments
                    satisfy argument constraints
```

The `pure` attribute is intended for the methods that do not change the state of their objects, but just return a new value. This attribute is either `false` (by default), or `true`.

Example

Consider the following code:

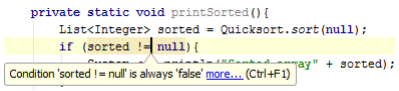
```
private static void
                    printSorted(){
                    List <Integer> sorted
                    = Quicksort.sort(null);
                    if (sorted != null){
                    System.out.println("Sorted array" + sorted);
                    }
                    }

                    public static <T extends
                    Comparable<T>> List<T> sort(List<T>
                    list)
                    {
                    if(list != null){
                    List<T> copy
                    = new ArrayList<T>(list);
                    sort(copy);
                    return copy;
                    }
                    else {
                    return null;
                    }
                    }
```

IntelliJ IDEA doesn't complain, because it doesn't know that a null input yields a null output.

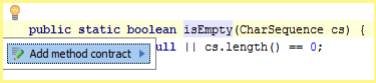
Let's decorate the `sort()` method with `@Contract` annotation, specifying that null inputs yield null outputs.

```
@Contract("null -> null")
public static <T extends Comparable<T>> List<T> sort(List<T> list)
{
    if (list != null){
        List<T> copy = new ArrayList<T>(list);
        sort(copy);
        return copy;
    }
    else {
        return null;
    }
}
```

IntelliJ IDEA immediately recognizes that `if` statement is extraneous, and reports about the condition that is always false:



Tip IntelliJ IDEA suggests a quick fix for the methods of the library classes:



@ParametersAreNonnullByDefault Annotation

– Overview
– Example

Overview

Annotation @ParametersAreNonnullByDefault gives the developer an option to define that for a given class or package all the elements (methods, parameters, fields and variables) have `@NotNull` semantic, unless they are explicitly annotated with the `@Nullable` annotation.

This is done by adding annotation `@javax.annotation.ParametersAreNonnullByDefault` to the entire package, class, or method.

To use `@javax.annotation.ParametersAreNonnullByDefault` annotation, make sure that `jsr305 jars` are added to the module libraries.

Example

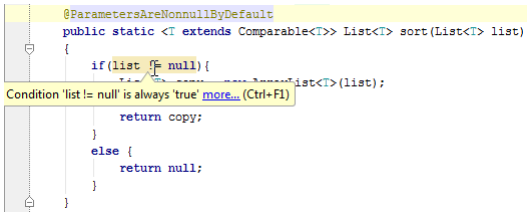
Consider the following code:

```
public static <T extends
                    Comparable<T>> List<T> sort(List<T>
                    list)
                    {
                    if(list != null){
                    List<T> copy
                    = new ArrayList<T>(list);
                    sort(copy);
                    return copy;
                    }
                    else {
                    return null;
                    }
                    }
```

IntelliJ IDEA doesn't complain.

Let's decorate the `sort()` method with `@ParametersAreNonnullByDefault` annotation, which means that all the symbols of this method behave as if they were @NotNull.

IntelliJ IDEA immediately recognizes that `if` statement is extraneous, and reports about the condition that is always true:



However, if you decorate the parameter of the method `sort()` as nullable, you will see no inspection messages:

```
@ParametersAreNonnullByDefault
public static <T extends Comparable<T>> List<T> sort(@Nullable List<T> list)
{
    if(list != null){
        List<T> copy = new ArrayList<T>(list);
        sort(copy);
        return copy;
    }
    else {
        return null;
    }
}
```

Annotating Source Code Directly

Your source code can contain various annotations. For example, `@Nullable` and `@NotNull` annotations are used, when IntelliJ IDEA supposes that a certain element can become `Null`. The `@NonNls` annotation is used to ignore the hard-coded string literals.

To use annotations

1. Make sure that `annotations.jar` that can be found in the `lib` folder under the IntelliJ IDEA installation, is added to the module dependencies.
   For `@javax.annotation.ParametersAreNonnullByDefault` annotations, `javax` should be added to the module libraries.

   > Tip You can also use annotations from JSR-305 and FindBugs. In this case, make sure they are added to the classpath.
   >
   > You can configure @Nullable, @NotNull and @Contract annotations in the Inspections page of the Settings dialog (Constant conditions and exceptions - Configure annotations).

2. Introduce the desired annotation before the package/class/field/variable/method/method parameter declaration.

External Annotations

Consider a situation, when one needs to annotate symbols with @Nullable/@NotNull or @NonNls , but to avoid annotations in the source code. This may be the case when a team works on a project, using different IDEs, for example, IntelliJ IDEA and Eclipse; or when working with library classes.

This section provides brief description of external annotations. To learn how to enable, configure and create external annotations, refer to the section Using External Annotation .

IntelliJ IDEA suggests using external annotations that are stored outside of the source code. Information about annotated symbols is stored in XML files, rather than in the source code. Each entry maps a method parameter or return value to the desired annotation class.

Such file has the name `annotations.xml` and resides in the path that depends on the origin of the specific annotation. If an annotation pertains to the SDK, configured for your project, the path is to be defined in the SDK settings. If an annotation pertains to a module, the path should be defined in the module settings.

Inferring Nullity

IntelliJ IDEA makes it possible to analyse the source code for the elements that can become null, and annotate them, provided that annotations are available in the project sources.

To automatically annotate nullable and non-null elements

1. Make sure that `annotations.jar` is added to your project. If it is not the case, IntelliJ IDEA will suggest to configure the annotations first. This can be done either manually (File | Settings | Project Settings - Inspections - Probable bugs - Constant conditions and exceptions), or automatically.
2. On the main menu, choose Analyze | Infer Nullity.
3. In the Specify Infer Nullity Scope dialog box, do the following:
   - Select the scope where you want to infer nullity: the entire project, the current file, etc.
   - If you want to perform nullity analysis for the test sources as well, and annotate local variables, select the corresponding check boxes.

   Click OK. IntelliJ IDEA adds import statement for annotations if required, and annotates parameters and variables.

Example

Consider the following code:

```
public Color myMethod(){
                    Color color = null;
                    return color;
                    }
```

Infer nullity for this code, with the check box Annotate local variables selected. IntelliJ IDEA annotates the method and local variable with the `@Nullable` annotation:

```
@Nullable
                    public Color myMethod(){
                    @Nullable Color color = null;
                    return color;
                    }
```

However, if in the initial code the variable is initialized with some value, rather than null, the method and the local variable will be annotated with the `@NotNull` annotation:

```
@NotNull
                    public Color myMethod(){
                    @NotNull Color color = new Color(255);
                    return color;
                    }
```

Using External Annotations

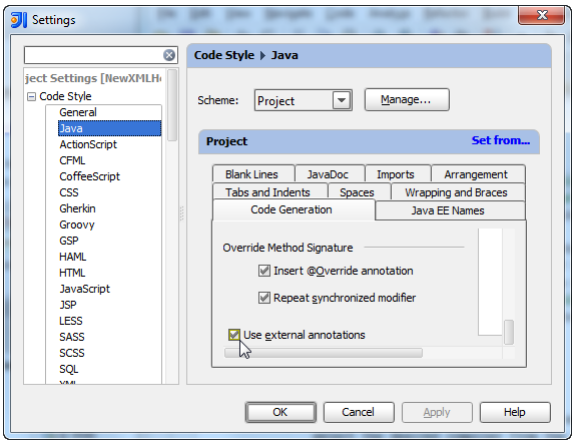
If you want IntelliJ IDEA to use external annotations , you have to enable this facility, and specify the directories, where the external annotations will be stored.

This section describes how to:

- Enable external annotations .
- Configure paths to the external annotations on the SDK or module level.
- Apply external annotations.

Enabling external annotations

1. In the Code Generation section of the Java code style page, select the check box Use external annotations.



2. Apply changes and close the dialog box.

Defining the path to external annotations on the SDK level

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. In the Platform Settings section, click SDK.
3. In the SDK page, select Annotations tab.
4. Click Add button and specify the desired path in the Select Path dialog. You can create as many annotation paths as you like.
5. Apply changes and close the dialog box.

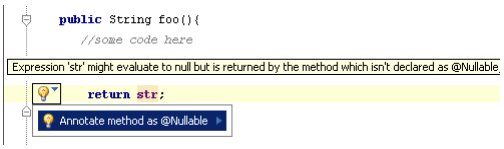
Defining the path to external annotations for a module

1. On the context menu of a project, choose Module Settings. Project Structure dialog is displayed on the Modules page .
2. Select Paths tab.
3. In the External Annotations section, click Add and specify the desired path in the Select Path dialog . You can create as many annotation paths as required.
4. Apply changes and close the dialog box.

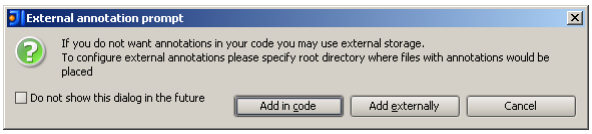
Annotating a symbol with an external annotation

Follow these general steps:

1. If code inspection alerts you about the necessity to annotate a symbol, press `Alt+Enter` to show the list of intention actions, and select the appropriate Annotate command, for example:



2. In the dialog box that opens, click the Add externally button.



Note This dialog appears if external annotations are enabled . Otherwise, annotations are added to the source code by default.

Documenting Source Code in IntelliJ IDEA

In this part:

- Documenting Source Code in IntelliJ IDEA
  - Basics
- Enabling Creation of Documentation Comments
- Creating Documentation Comments
- Generating JavaDoc Reference for a Project

Basics

IntelliJ IDEA provides convenient features for creating documentation comments.

Documentation comments in your source code are available for the Quick Documentation Lookup and open for review on pressing `Ctrl+Q`.

In Java files, IntelliJ IDEA creates stubs of documentation comments on typing the opening tag and pressing `Enter`.

Documentation comments can be created in accordance with the RDoc or YARD syntax.

If this feature applies to the methods, `@param` tags are created for each parameter declared in the method signature, `@return` tag is created if the method is not void, and `@throws` tags are created for each exception statement.

When you create additional tags , IntelliJ IDEA provides code completion that suggests the possible tag names.

If a certain tag has multiple values, the same code completion provides the list of available values. Smart type code completion suggests the list of classes that are appropriate for the specific exception.

Enabling Creation of Documentation Comments

Warning! Note that this section refers to JavaScript, Java and the other languages that have special beginning of documentation comments.

To enable or disable generation of documentation comments

1. Open the Settings dialog box , expand the Editor node, then expand General node, and click the Smart Keys page.
2. In the Enter section, select or clear Insert documentation comment stub check box.

Creating Documentation Comments

In this section:

– Creating documentation comments for a method or function
– Creating tags
– Creating and fixing doc comments
– Clickable comments

Creating documentation comments for a method or function

To create a documentation comment for a method or function

1. Place the caret before the declaration.
2. Type the opening block comment `/**`, and press `Enter`.
3. Add meaningful description of parameters and return values.

Please note the following:

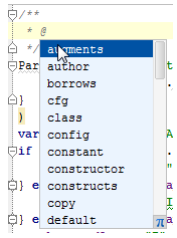
– IntelliJ IDEA checks syntax in the documentation comments and treats it according to the Error settings.
– If the entered text contains HTML tags, the closing tag will be automatically added after typing `>`, provided that this behavior is enabled in the editor settings.
– When typing in a documentation comment, the caret automatically moves to an expected position. For example:

```
/**
 * <ul>
 *     <li>[caret]
 *         [caret after Enter]
 *     </li>
 * </ul>
 */
```

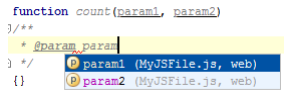
Creating tags

To create tags in a documentation comment block

1. In a comment block, select the desired empty line and type `@` character.
2. Press `Ctrl+Space`, or just wait for Code Completion to display the suggestion list:



3. Select a tag from the suggestion list. For example, you can specify the parameters type, or return type.
4. If a certain tag has several values, press `Ctrl+Space` after the tag, and select the desired value from the suggestion list. For example, IntelliJ IDEA suggests to select the desired parameter name.



Creating and fixing doc comments

Warning! Note that this section refers to JavaScript, Java and the other languages that have special beginning of documentation comments.

Documentation comment can be created with the dedicated action Fix Doc Comment. It can be invoked by means of Find Action command.

Press `Ctrl+Shift+A`, with the caret somewhere within a class, method, function, or field, which should be documented, and enter the action name Fix Doc String. The missing documentation stub with the corresponding tags is added. For example:

```
/**
                    *
                    * @param userInput
                    * @return
                    */
                    static boolean processRepeatConversion (@NotNull String userInput) {
                    boolean repeatConversion = false;
                    if (((userInput.equals("y")) || (userInput.equals("Y")))) {
                    repeatConversion = true;
                    }
                    return repeatConversion;

                    }
```

The next case lays with fixing problems in the existing documentation comments.

For example, if a method signature has been changed, IntelliJ IDEA highlights a tag that doesn't match the method signature, and suggests a quick fix.

For JavaScript, IntelliJ IDEA suggests an intention action UpdateJSDoc comment. You can also press `Ctrl+Shift+A`, and type the action name:

Tip The action Fix doc comment has no keyboard shortcut bound with it. You can configure keyboard shortcut of your own.

Clickable comments

If a documentation comment contains a hyperlink, you can make it clickable.

To do that, do one of the following:

– Keep the `Ctrl` key pressed and hover your mouse pointer over the hyperlink:

– Press `Ctrl+B`.

Generating JavaDoc Reference for a Project

IntelliJ IDEA provides a front-end to the standard JavaDoc utility for generating JavaDoc reference for your project. This feature is available from the editor and from the project tool window.

To generate project documentation

1. On the main menu, choose Tools | Generate JavaDoc. Generate JavaDoc dialog is opened.
2. In the Generate JavaDoc dialog, specify the following options:
   - Select scope (whole project or a certain project with subpackages).
   - Specify the output directory, where the generated documentation will be placed.
   - Use the slider to define the level of visibility of members to be included in the generated documentation.
3. Specify the other JavaDoc options. Refer to the topic Generate JavaDoc Dialog for description of controls.
4. Click OK.

Viewing Reference Information

IntelliJ IDEA facilitates quick and easy access to the API documentation, which you can view immediately in the editor or in an external browser. To gain access to the API documentation from within your project, make sure to attach archives or directories that contain sources of the library classes.

This section describes how to use the view parameter information feature, display context information or JavaDoc references, and see definitions of the symbols:

- Viewing Definition
- Viewing Inline Documentation
- Viewing External Documentation
- Viewing Method Parameter Information

Viewing Definition

On this page:

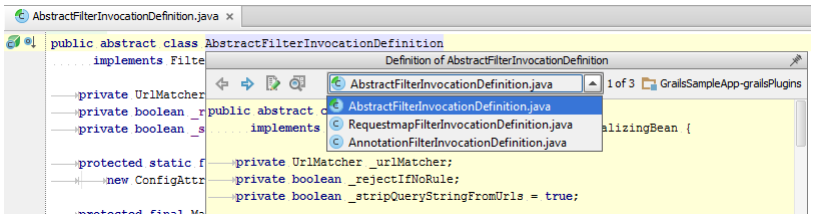
– Basics

– Viewing the definition of a symbol at caret

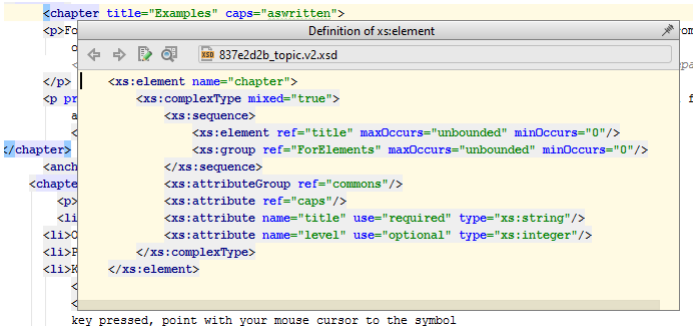
– Toolbar of the quick definition lookup

Basics

Quick Definition Lookup makes it possible to view definition of a symbol (tag, class, method/function, field, etc.) in a pop-up window.



For markup languages, IntelliJ IDEA retrieves definitions of symbols from the specified DTD or schema. For details, see Markup Languages and StyleSheets .



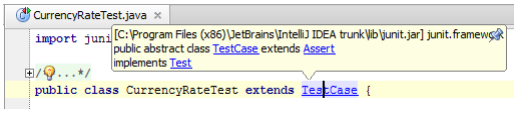
Viewing the definition of a symbol at caret

Do one of the following:

– On the main menu, choose View | Quick Definition.

– Press `Ctrl+Shift+I` .

– Keeping the `Ctrl` key pressed, point with your mouse cursor to the symbol of interest, so that it turns to a hyperlink, with the definition of the symbol displayed in a tooltip. Clicking this hyperlink results in opening the respective definition page in the editor.
Quick definition tooltip shows hyperlinks to the symbols involved.



When you move your mouse pointer within the tooltip, a pin button appears. If you pin the tooltip, documentation for the symbol at caret is displayed in the Documentation Tool Window .

Toolbar of the quick definition lookup

Use the icons on the toolbar of the pop-up window to navigate to the source code of the definition and view its usages.

Icon	Keyboard shortcut	Action
	`Shift+Alt+Left` `Shift+Alt+Right`	Navigate to the previous/next screen in the definition pop-up window after using hyperlinks in the definition. **Note** On an OS X computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.
	`F4`	Open the source code of the definition for editing, and close the quick definition lookup window. If IntelliJ IDEA cannot find the appropriate sources in your project, it will try to decompile the code. In this case the JetBrains Decompiler dialog with legal information is displayed.
	`Ctrl+Enter`	Open the source code of the definition, and preserve the quick definition lookup window opened. If IntelliJ IDEA cannot find the appropriate sources in your project, it will try to decompile the code. In this case the JetBrains Decompiler dialog with legal information is displayed.

Viewing Inline Documentation

On this page:

– Basics

– Viewing quick documentation

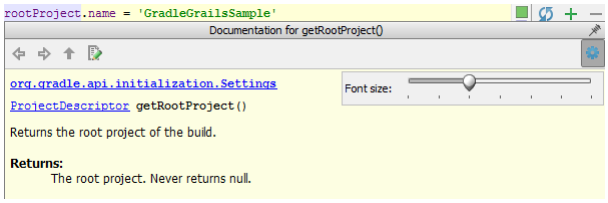
– Toolbar of the quick documentation lookup

Basics

Quick Documentation Lookup helps you get quick information for any symbol or just method signature information, provided that this symbol has been supplied with documentation comments in the applicable format.

IntelliJ IDEA recognizes inline documentation created in accordance with Javadoc markup.

In this case, such documentation is properly rendered in the Quick Documentation Lookup window:



For markup languages , IntelliJ IDEA retrieves reference from the language specification according to the Document Type setting.

The URLs and e-mail addresses specified in the documentation comments for methods are also properly rendered. Clicking a hyperlink opens the corresponding URL in an external browser; clicking an e-mail address opens the default mail client.

Viewing quick documentation

To view documentation for a symbol at caret, do one of the following

– On the main menu, choose View | Quick Documentation Lookup.

– Press `Ctrl+Q`.

– Provided that the check box Show quick doc on mouse move in the editor settings is selected, just move your mouse pointer over the desired symbol.

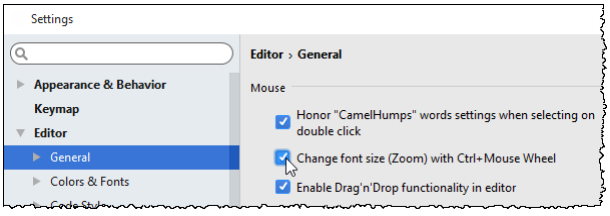
When you explicitly invoke code completion, then quick documentation for an entry selected in the suggestion list can be displayed automatically. The behavior of quick documentation lookup is configured in the Code Completion page of the Settings/Preferences dialog.

Tip Sequential pressing `Ctrl+Q` toggles the focus of the Quick Documentation pop-up window and Documentation Tool Window .

To change the font size of quick documentation, do one of the following

– Click in the upper-right corner of the quick documentation window, and move the slider.

– Rotate the mouse wheel while keeping the `Ctrl` key pressed.
Note that for this feature to work, you have to enable it in the General page of the editor settings.



Refer to the section Zooming in the Editor for details.

Toolbar of the quick documentation lookup

The Quick Documentation Lookup window helps navigate to the related symbols via hyperlinks, and provides a toolbar for moving back and forth through the already navigated pages, change font size, and viewing documentation in an external browser.

When pinned, the Quick Documentation Lookup turns into Documentation Tool Window , with the corresponding sidebar icon, and more controls.

Icon	Shortcut	Description
	`Left` or `Right`	Switch to the previous or next documentation page (e.g. after using hyperlinks). **Note** On an OS X computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.

	Shift+F1	View external documentation in the default browser.
	F4	Switch to the item (e.g. source) that corresponds to the documentation page currently shown.
		Turn the Auto-update from source option on or off. When the option is on, the information in the tool window is synchronized with your navigation in the editor and other places in the UI.
		Click this icon to show font size slider. Move the slider to increase or decrease the font size in the quick documentation window as required.

Tip For information on retrieving inline documentation in the JavaScript or PHP context, refer to sections Viewing JSDoc Comments and Viewing PHPDoc Comments respectively.

Viewing External Documentation

External documentation makes it possible to get additional information for the symbols at caret. In contrast to the quick documentation , this feature shows the documentation in an external browser, which helps study the symbol in more detail, navigate to related symbols, and retain the information for further reference.

External documentation becomes available for viewing when properly configured in the module structure . For example, in the module paths , you can add a path to a JavaDoc file, or a link to documentation; or specify documentation URL for a library .

To view documentation for a symbol at caret in an external browser, do one of the following:

- On the main menu, choose View | External documentation.
- Press `Shift+F1` .
- While in the Quick Documentation Lookup window , click .

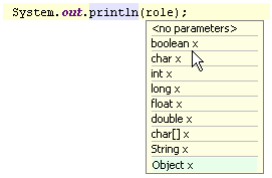
Viewing Method Parameter Information

This section describes how to configure the View Parameter Information behavior and use this functionality.

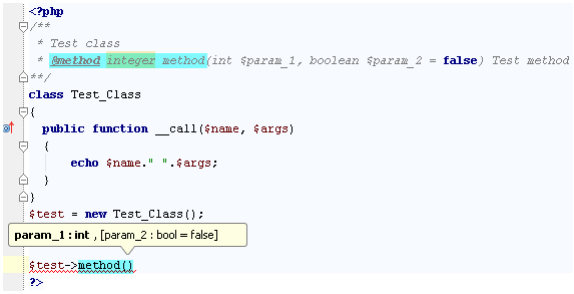
To view the method parameters

1. Place the caret anywhere within the call of the desired method or function.

2. Choose View | Parameter Info on the main menu or press `Ctrl+P`.



Parameter information for methods defined through the `@method` phpDocumentor tag is also available:



To configure the behavior of the view parameter information functionality

1. Open the IDE Settings and click Code Completion below the Editor node.

2. In the Parameter info section, define the following options:
   - To have a complete method or function signature shown rather than a list of required types, select the Show full signatures check box.
     Make sure to include the required third-party libraries in the project source path. Otherwise, names of the parameters will not be displayed.
   - To have the list of parameter types for the called method or function shown automatically after a certain delay, select the Auto pop-up (in ms) check box and specify the time period in milliseconds.

Using Language Injections

In this section:

- Using Language Injections
  - Introduction
  - Prerequisites
  - Injecting a language into the source code and cancelling language injection
  - Moving injections between the IDE and the project levels
  - Using comments
- Opening and Closing Language Injections in the Editor

Introduction

IntelliJ IDEA makes it possible to work with islands of different languages embedded in the source code. You can inject other languages into string literals. This can be done within the source code written in most (but not all) of the supported languages (Java, JavaScript, Groovy, Python, Ruby, XML, PHP, and CSS for IntelliJ IDEA Ultimate Edition). The typical examples are HTML fragments injected into JavaScript code, SQL statements in Java or XML, and so on.

When the editor recognizes a string as a language injection:

- Syntax and error highlighting and coding assistance are extended to this string.
- You can open and modify it in a separate tab in the editor , as if you were working with the source code in the corresponding language.
  To open an injection in the editor, use the Edit <Language> Fragment intention action .

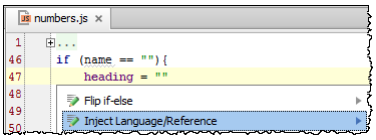
Prerequisites

Before you start working with language injections, make sure that the IntelliLang plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

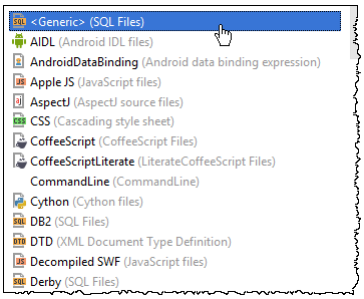
Injecting a language into the source code and cancelling language injection

To tell IntelliJ IDEA that certain text should be treated as an embedded source code fragment, you can use:

- The Inject Language/Reference intention action , which is applied to a particular string:



After that, you have to select the desired language from the menu.



Note that if you use this method, it's possible that the string literal will stay marked as a language injection only within a limited period of time. That is, IntelliJ IDEA, at a certain moment, may "forget" that the corresponding literal is a language injection. The period of the injection "persistence" will depend on the language, context and the modifications that you make in other parts of your source code.

- A dedicated @Language annotation in Java source code, for example, `@Language("JavaScript")` or `@Language("HTML")` :

```
public class Solver {
    @Language("SQL")
    String s = "select AlbumId from Album";
}
```

- Comments such as `//language=<language_ID>` , `/*language=<language_ID>*/` , `#language=<language_ID>` , `<!--language=<language_ID>-->` , `--language=<language_ID>` .
  For more information, see Using comments.
- The Language Injections page of the Settings dialog . By creating new language injection configurations on that page, you can specify that a certain method parameter, text in an XML tag, or XML attribute should always be treated as an embedded source code fragment in another language.

> **Warning!** IntelliJ IDEA comes with a set of predefined injection configurations which is quite sufficient to ensure high productivity and comfortable environment. Therefore it is strongly recommended that you use the predefined injection configurations and avoid creating new ones.

For example, SQL injection in JavaScript code will look as follows:

```
var i;
                    for (var j=1;j<10;j++){
                    var s= "SELECT * FROM A WHERE
                    B=C";
                    }
```

To cancel language injection, choose the intention action Un-inject Language/Reference.

Moving injections between the IDE and the project levels

User-defined injections are assigned either to the IDE or the project level. The IDE-level injections are available in all of your projects. The project-level injections can be used only in the project in which they are defined.

To move the injections between the IDE and the project levels, use the Move to Project/IDE Scope icon on the Language Injections page of the Settings/Preferences dialog .

The built-in injections are the IDE-level ones, and their scope cannot be changed.

Using comments

Here is the syntax to be used when declaring language injections by means of comments:

```
//language="<language_ID>" prefix="<prefix>"
                    suffix="<suffix>"
```

`<language_ID>` is an ID of the injected language. The `prefix` and `suffix` are optional.

If specified, the prefix is added before the string literal and the suffix - after. The prefix, the string literal itself and the suffix are added all together to form a single injected language fragment.

Quotation marks are used if there are spaces within the language ID, prefix or suffix. Only double quotes can be used (`"`).

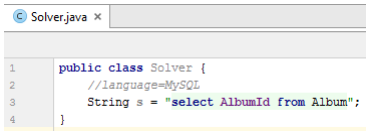
In most of the cases the language ID is intuitive, e.g. CSS, HTML, SQL, MySQL, DB2, Oracle, PostgreSQL and so on. If you are not sure about the language ID, use the suggestion list for the Inject Language intention action as the source of information. For example, type `""` in the editor and place the cursor between `"` and `"` .

Press `Alt+Enter` and select Inject Language. In the list that is shown, what precedes the opening parentheses are language IDs.

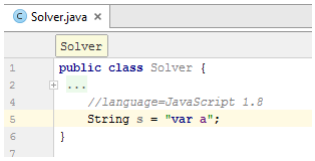
The comment should precede the string literal to which it applies and should be placed as close as possible to that literal.

The following examples illustrate the use of comments for language injections in Java code.

MySQL:



JavaScript 1.8:



Opening and Closing Language Injections in the Editor

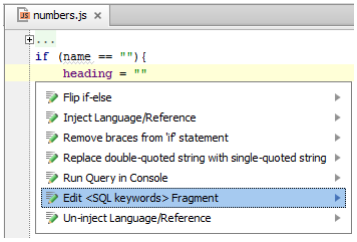
Language injections are opened in the editor by means of a dedicated intention action Open <Injected Language> Fragment.

To open a language injection in the editor:

1. In the editor, place the cursor within the string literal that represents the language injection of interest.

2. To show the available intention actions, do one of the following:
   - Click the yellow bulb.
   - Press `Alt+Enter`.

3. To select the Edit <Injected Language> Fragment option, do one of the following:
   - Click this option.
   - Use the `Up` and `Down` arrow keys to navigate to the option, and then press `Enter` to select it.



Tip You can open a number of language injections in separate editors, and copy code fragments from one editor to another. However, as soon as you start modifying the code in one of the editors, all the rest of the editors close automatically.

To close an editor for a language injection , you can use the same options as for closing files .

Internationalization and Localization Support

In this section:

- Internationalization and Localization Support
  - Overview
  - i18n-related features
- Properties Files
- Resource Bundle
- Resource Files
- Configuring Encoding for Properties Files
- Creating and Editing Properties Files
- Editing Resource Bundle
- Extracting Hard-Coded String Literals
- Ignoring Hard-Coded String Literals
- Recognizing Hard-Coded String Literals

Overview

i18n support with IntelliJ IDEA falls into the following major aspects:

- Internationalization, which involves extracting strings out of your source code and presenting them as properties that are further referenced in the source code.
- Localization, which means translating these properties into the target languages.

i18n-related features

IntelliJ IDEA provides helpful features that simplify working on software internationalization and localization issues. These features are:

- Individual encoding for files and directories.
- Dedicated file type for storing properties.
- Advanced editing assistance for properties files.
- Auto-detection of resource bundles .
- Code inspections related to internationalization issues; intention actions, and quick fixes.
- Dedicated editor for performing mass actions within resource bundles.

Properties Files

On this page:

- Basics
- Properties file features

Basics

The properties files are text files with the `.properties` extension, containing pairs of keys and values, that can be accessed and rendered in the UI.

These files are marked with the icon .

IntelliJ IDEA also recognizes properties files in XML format. They are marked with the icon .

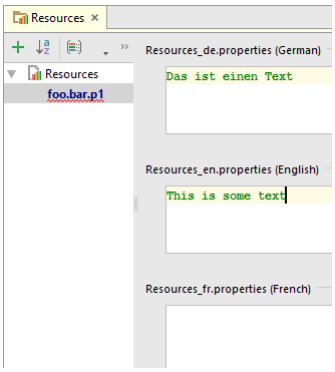
Properties file features

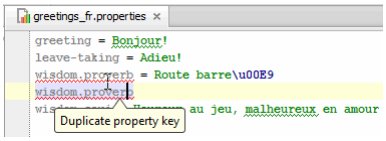
IntelliJ IDEA supports the following features for the properties files:

- Action for creating new resource bundles .
- Ability to work with XML-based properties files.
- Actions for combining and dissociating properties files.
- Syntax `key - delimiter - value` . IntelliJ IDEA uses `=` (equal sign) or `:` (colon) as a delimiter.
  For example, if you want a field label to read "Your name:", you might create a pair in a properties file like this:

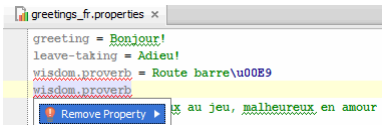
  `nameLabel=Your name:`

  `nameLabel: "Your name:"`
- Error highlighting for errors like missing locale records, or duplicate property keys and invalid escape sequences :

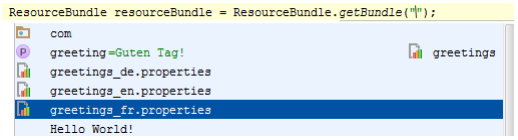




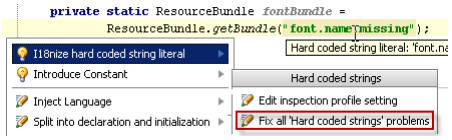
- Inspection and quick fix for detecting and removing duplicate keys in properties files:



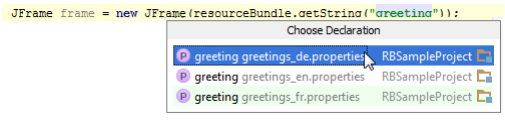
- Referencing properties from Java files and Ant build files.
  - Code completion for property keys:



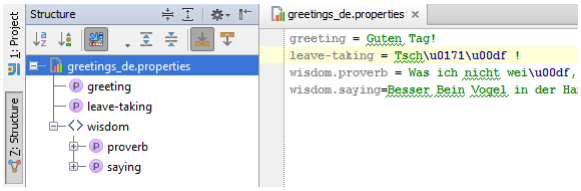
  - Inspection and quick fix for detecting and creating the missing property keys and values:



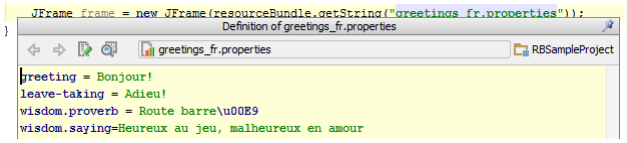
  - Goto Declaration `Ctrl+B` to navigate from a reference to a key to its declaration:



  - Finding usages of the currently selected property.
  - Refactoring for properties includes Rename, Move, Copy, Safe Delete, and Migrate.
- Structure view for properties files:



– Quick Definition Lookup for properties files: if an action calls a property, you can view the property definition by pressing `Ctrl+Shift+I`:



Resource Bundle

On this page:

– Basics
– Creating resource bundles
– Combining or dissociating properties

Basics

Resource bundle is a set of properties files that have same base name with different language-specific suffixes. A resource bundle contains at least two properties files with similar base name, for example `file_en.properties` and `file_de.properties`.

IntelliJ IDEA recognizes resource bundles, and marks them with the icon .

IntelliJ IDEA supports the following features for the resource bundles:

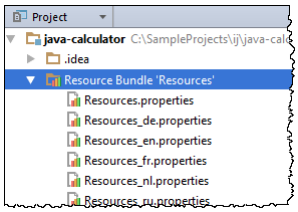
– Action for creating new resource bundles .
– Actions for combining and dissociating properties files.
– Dedicated Resource Bundle Editor .
– Error highlighting for the missing properties or values.
– Refactoring for resource bundles includes Rename , Move , Copy and Safe Delete .

Creating resource bundles

To create a new resource bundle, follow these steps

1. In the Project tool window, choose the directory where the new resource bundle should be created.
2. Do one of the following:
   – Press `Alt+Insert`.
   – On the context menu of the selection, choose New | Resource Bundle
   – On the main menu, choose File | New | Resource Bundle
3. In the dialog box that opens, do the following:
   – Specify the base name of the resource bundle.
   – If necessary, select the check box Use XML-based properties files.
   – Add the required locales. To do that, click + and type the comma-separated suffixes of the required locales.
   – Click OK when ready.

The new node Resource Bundle '<base name>' appears in the Project Tool Window :



Combining or dissociating properties

By default, when a new resource bundle is created, it shows joined. You can dissociate it and show the properties files only.

To dissociate a resource bundle

1. Right-click the resource bundle you want to dissociate.
2. On the context menu, choose Dissociate Resource Bundle <base name>.

To combine several properties files into a resource bundle

1. Select the properties files to be combined.
2. Right-click the selection.
3. On the context menu, choose Combine to Resource Bundle.
4. Specify the base name of a resource bundle.

Resource Files

Resources include properties files , images, DTDs, and XML files. These files are located under the Classpath of your application, and are usually loaded from the Classpath by means of the following methods:

- `ResourceBundle.getBundle()` for the property files and resource bundles
- `loadResourceAsStream()` for icons and other files

When building an application, IntelliJ IDEA copies all resources into the output directory, preserving the directory structure of the resources, relative to the source path. The following file types are recognized as resources by default:

- `.properties`
- `.xml`
- `.html`
- `.dtd`
- `.tld`
- `.gif`
- `.png`
- `.jpeg`
- `.jpg`

The pattern of recognized resource files is represented as a regular expression and configurable in the Compiler dialog .

Using Resource pattern setting, you can add your own file extensions and create custom list of resources.

Configuring Encoding for properties Files

Standard Java API is designed to use ISO 8859-1 encoding for the properties file.

To use other encodings, feed them as escape sequences and Unicode. The other option is to define the default encoding for `.properties` files on the project level and use different API that can read properties files in the encoding you have defined.

To configure default encoding for properties files

1. Open the Settings dialog box and click File Encodings .

2. Do one of the following:
   - To have the special mode turned on when symbols are stored in a properties file as escape sequences but displayed as normal letters, check the option Transparent native-to-ascii conversion. This option is helpful when the properties files are encoded in ISO 8859-1. It is recommended to use this approach, if you don't have any special reasons to change encoding.
   - In the field Default encoding for properties files, select the encoding that will be used for all properties files in project.

Creating and Editing properties Files

On this page:

– Basics
– Creating locales
– Editing locales

Basics

The properties files are used to store localization strings. IntelliJ IDEA enables you to create new properties files, and detects the existing ones.

Creating locales

To create properties files

1. Right-click a directory where you would like to create a properties file.
2. On the context menu of the target directory, choose New | File.
3. In the New File dialog box, type the file name with the corresponding extension, and click OK. Note that you can also create nested directories under `locales`, and add new properties files there.

Tip If you create several files with the same name and different locale suffixes (for example, `myProperty_en`, `myProperty_fr` etc.), such properties files will be recognized as a resource bundle .

Editing locales

While editing properties files, use the following techniques:

– Syntax highlighting: Keywords, delimiters, values and comments are highlighted. You can configure the color scheme in the Colors and Fonts page of the Settings/Preferences dialog.

– Code inspection detects unused strings and suggests to fix the problem immediately:

– Expand word: Start typing, and press `Alt+Slash` to go through the existing strings.

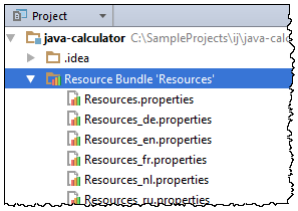
Editing Resource Bundle

On this page:

- Resource bundle editor basics
- Invoking properties editor for a resource bundle
- Editing property keys and values
- Tips and tricks

Resource bundle editor basics

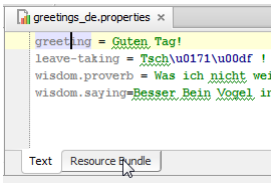
Once you create several `.properties` files with the same name, differing by locale suffix, IntelliJ IDEA automatically recognizes them and groups in the Project view into a Resource Bundle.



Invoking properties editor for a resource bundle

To invoke properties editor for a resource bundle, do one of the following:

- In the Project tool window, right-click a resource bundle and choose Jump to Source.
- Select a resource bundle in the Project tool window, and press `F4`.
- Open for editing a `.properties` file that is a part of a bundle, and at the lower edge of the editor, click Resource Bundle tab:



Editing property keys and values

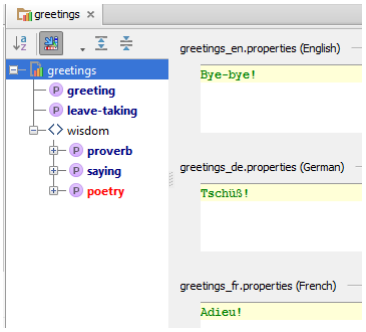
To edit property keys, follow these general steps

1. Open for editing the desired `*.properties` file.
2. Add, change, or delete keys as required. The changes are reflected in the Resource Bundle editor.

For changing values, use the resource bundle editor that enables you to edit an entire set of property files at a time. IntelliJ IDEA takes care of creating respective records in each file of the bundle.

To edit a property value

1. Select property key in the left pane of the resource bundle editor.
2. In the target locale frame, edit the value as required. The respective `.properties` file will change accordingly.



Tips and tricks

- The properties missing values, or omitted in one of the `.properties` files are red-highlighted.
- All escaped characters in the `*.properties` files in the format `\uXXXX`, are displayed in the resource bundle editor as un-escaped unicode literals.
  Vice versa, if a non-ASCII character is entered in the resource bundle editor, it is reflected in the underlying `*.properties` file as a corresponding escaped character in the format `\uXXXX`.

  For example, if the `*.properties` file contains a property value

```
Was ich nicht wei\u00df, macht mich nicht hei\u00df
```

then the resource bundle editor will show

```
Was ich nicht weiß, macht mich nicht heiß
```

Resource bundle editor itself does not perform any conversion. To have escape sequences properly resolved in properties files, select the check box Transparent native-to-ascii conversion in the File Encoding page of the Settings/Preferences dialog.

- It is possible to encode non-ascii symbols using both upper- and lower-case hex symbols (e.g. `'\u00E3'` vs `'\u00e3'`). Upper case is used by default. To use lower case, set 'idea.native2ascii.lowercase' property in the `bin/idea.properties` file to true.
  Refer to the section Tuning IntelliJ IDEA for details.

Extracting Hard-Coded String Literals

On this page:

- Introduction
- Extracting string literals
  - Extracting string literals using ResourceBundle
  - Extracting string literals using custom resource bundle class

Introduction

Having enabled code inspection that highlights hard-coded string literals, you can proceed with extracting these literals into your properties files. For this purpose, IntelliJ IDEA provides special intention action i18nize hard coded string literal.

This section considers two possible ways of accessing resource bundle:

- Using the java.util.ResourceBundle utility class
- Using custom utility class

Extracting string literals

Extracting string literals using ResourceBundle

To extract a string literal using java.util.ResourceBundle

1. Specify resource bundle that will be used to store the extracted literals. In particular, you can add the following statement to your source code:

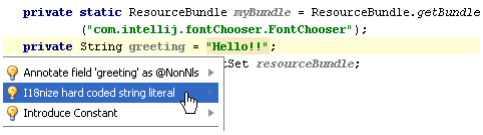
```
private static ResourceBundle
                    <field name> =
                    ResourceBundle.getBundle("<bundle name>");
```

   For example:

```
private static ResourceBundle
                    myBundle
                    =
                    ResourceBundle.getBundle("com.intellij.fontChooser.FontChooser");
```

   Note You can skip this step and specify the desired resource bundle expression immediately in the I18nize Hard Coded String Literal dialog box.

2. Click the highlighted string, press `Alt+Enter`, and in the list of intention actions choose i18nize hard coded string literal:



3. In the I18nize Hard Coded String Literal dialog box specify the target properties file, the property key and value, and the resource bundle expression.
   If the ResourceBundle field has been declared in the source code (as shown in the step 1), IntelliJ IDEA suggests its name by default. If you haven't declared this field in the source code yet, you can still define the desired expression immediately in the dialog box. To do that, enter a valid expression of the ResourceBundle type in the Resource bundle expression field.

   Note that basic code completion works in this field. Type class name, and press `Ctrl+Space` after period to select method:

   Choose the desired method from the suggestion list, and press `Ctrl+Space` once more to fill in the parentheses:

   After that, type the package and resource bundle name in quotes:

4. Click OK. The line with hard-coded string literal is replaced. For example, if the resource bundle has been declared in the source code, the following line will be created:

```
private String greeting = myBundle.getString("hello");
```

   If the resource bundle has been defined in the dialog, the result will be:

```
private String greeting = ResourceBundle.getBundle
    ("com.intellij.fontChooser.FontChooser").getString("hello");
```

Extracting string literals using custom resource bundle class

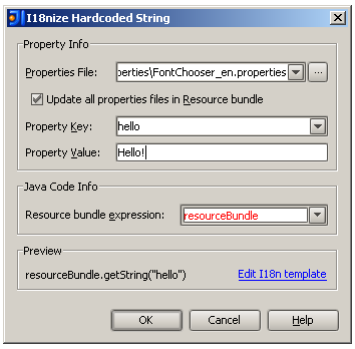
To extract a string literal using custom resource bundle class

1. Make sure that `redist/annotations.jar` archive that resides under your IntelliJ IDEA installation, is added to the module dependencies .

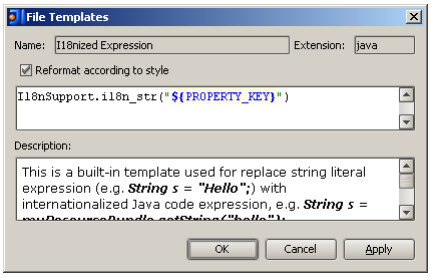
2. Create a new class in your project, and type the following code:

```
import org.jetbrains.annotations.PropertyKey;
                    import org.jetbrains.annotations.NonNls;
                    import java.util.ResourceBundle;
                    import java.text.MessageFormat;
                    public class I18nSupport {
                    @NonNls
                    private static final ResourceBundle
                    bundle = ResourceBundle.getBundle ("com.intellij.FontChooser");
                    public static String i18n_str
                    (@PropertyKey(resourceBundle
                    ="com.intellij.FontChooser")
                    String key,Object...
                    params){
                    String value =bundle.getString(key);
                    if (params.length
                    >0) return MessageFormat.format(value,
                    params);
                    return value;
                    }
                    }
```

3. In a class that contains hard-coded string, click the highlighted string, press `Alt+Enter`, and in the list of intention actions choose i18nize hard coded string literal. The I18n-ize String Literal dialog box shows that resource bundle expression is missing:



4. Click Edit i18n template link. In the File Template dialog box, change the I18nized Expression to point to the method of your custom resource bundle class:



Click OK to save the updated template and close the dialog box.

> **Warning!** This change is global, and affects all projects!

5. In the Preview section of the I18n-ize String Literal dialog box, see the suggested substitution, and click OK. The source code changes:

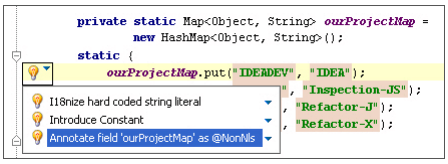
```
private String greeting = I18nSupport.i18n_str("hello");
```

Ignoring Hard-Coded String Literals

Sometimes you might want to ignore the hard-coded literals. To do that, use *Not requiring internationalization* annotation.

To ignore a certain hard-coded literal

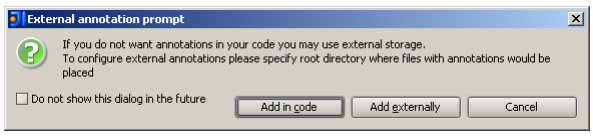
1. Press `Alt+Enter` to show intention actions for the string literal:



2. Select Annotate as @NonNls from the suggestion list.

3. In the Select Path dialog that opens, specify the location where `annotations.xml` file will be stored.

Alternatively, you can add `@NonNls` annotation:



If you have selected the check box Do not show this dialog in the future, you can still choose annotation style in the Settings dialog (Code Style | Java - Code Generation).

Recognizing Hard-Coded String Literals

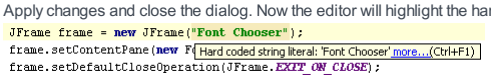
Your source code can contain hard-coded literals which you would like to recognize and further extract to properties . To highlight hard-coded literals in the editor, use Internationalization code inspections.

To enable recognizing hard-coded string literals

1. Open the Settings dialog .
2. In the Project Settings, select Editor | Inspections to open Inspections dialog box.
3. Select the desired profile, and locate the node Internationalization issues.
4. Check the option Hard coded strings, which will cause the hard-coded string literals to be highlighted in the editor.

   Note Note that you can configure severity of this inspection and specify when the hard-coded strings should be recognized. For example, you can opt to skip the content of `toString()` method, or the literals without alphabetic characters.

5. Apply changes and close the dialog. Now the editor will highlight the hard-coded string literals, as shown below:



Navigating Through the Source Code

IntelliJ IDEA suggests various ways of navigation between the IDE components and within source code. IntelliJ IDEA's smart editor makes it possible to navigate across the source code using the code structure rather than plain scrolling. You can find your way in the source code using the method calls, declarations, errors, changes etc.

In this part:

- Navigating with Bookmarks
- Navigating Between Files and Tool Windows
- Navigating Between IDE Components
- Navigating Between Methods and Tags
- Navigating Between Test and Test Subject
- Navigating from Stacktrace to Source Code
- Navigating to Action
- Navigating to Braces
- Navigating to Class, File or Symbol by Name
- Navigating to Custom Folding Regions
- Navigating to Declaration or Type Declaration of a Symbol
- Navigating to Super Method or Implementation
- Navigating to File Path
- Navigating to Line
- Navigating to Next/Previous Change
- Navigating to Next/Previous Error
- Navigating to Recent
- Navigating to Navigated Items
- Navigating with Breadcrumbs
- Navigating with Favorites Tool Window
- Navigating with Navigation Bar
- Navigating with Structure Views

Navigating with Bookmarks

On this page:

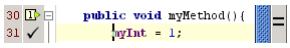
- Introduction
- Navigating within the current file
- Navigating across a project

Introduction

The editor of IntelliJ IDEA provides two types of bookmarks:

- Anonymous bookmarks, indicated by check signs in the left gutter. The number of anonymous bookmarks is unlimited.
- Bookmarks with mnemonics indicated by or icons in the left gutter. There can be only 10 numbered and 26 lettered bookmarks within a project.

All bookmarks are indicated with the black streaks in the marker bar:



Once created , the bookmarks enable you to easily jump to the desired location within a file, or across the entire project.

Navigating within the current file

To navigate through the bookmarks within the current file, do one of the following

- On the main menu, choose Navigate | Bookmarks | Next/Previous Bookmark. The order of visiting bookmarks depends on their order in the collection of bookmarks in the Bookmarks dialog .
- Click the black streak in the marker bar.

Navigating across a project

To navigate across a project using numbered bookmarks

- Use `Ctrl+Number` where the <number> corresponds to the desired bookmark.

To navigate among all bookmarks in a project, do one of the following

- On the main menu, choose Navigate | Bookmarks | Show Bookmarks, or press `Shift+F11`.
  In the Bookmarks dialog, select the target bookmark, and press `Enter`.

  For your convenience, the target code preview is shown in the right pane of the dialog box.
- In the Favorites tool window , select the desired bookmark in the Bookmarks list, and then double-click the bookmark entry, or press `F4`. The corresponding file opens in the editor, with the caret at the beginning of the bookmarked line.

Managing Bookmarks

This section describes how to:

- Create and delete bookmarks with mnemonics
- Toggle bookmarks
- View project bookmarks
- Delete bookmarks
- Change the order of bookmarks

To create a bookmark with mnemonics

1. Place the caret at the desired line of code in the editor.
2. Press `Ctrl+F11` (alternatively, choose Navigate | Bookmarks | Toggle Bookmark With Mnemonic on the main menu) , then press one of the keys 0-9 or A-Z.

To toggle an anonymous bookmark on the current line

- On the main menu, choose Navigate | Bookmarks | Toggle Bookmark.
- Press `F11`.

To view all bookmarks in a project

- On the main menu, choose Navigate | Bookmarks | Show Bookmarks.
- Press `Shift+F11`.

To delete bookmarks in a project

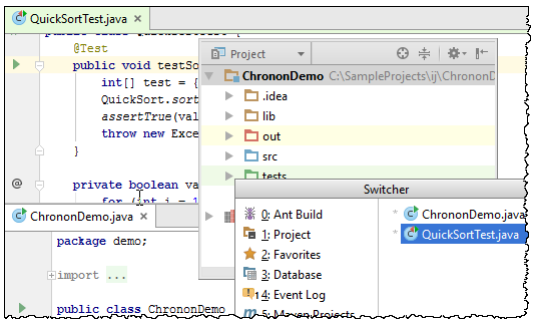
1. Open the Bookmarks dialog (Navigate | Bookmarks | Show Bookmarks, or `Shift+F11`).
2. Select bookmarks and press `Delete`.

To change the order of bookmarks

1. Open the Bookmarks dialog (Navigate | Bookmarks | Show Bookmarks, or `Shift+F11`).
2. Select the desired bookmark.
3. Use Move Up (, `Ctrl+Up`) and Move Down (, `Ctrl+Down`) buttons to shift the bookmark in the desired direction.

Navigating Between Files and Tool Windows

IntelliJ IDEA suggests a handy way to switch between files opened in the editor, split editor tabs, and tool windows (docked or floating). This is similar to the application switchers in the various operating systems. The switcher consists of two columns: the left one displays the list of tool windows, while the right one displays the list of files currently opened in the editor.



To switch between open files and tool windows

1. Press `Ctrl+Tab`.
2. Keeping `Ctrl` pressed, use the following keys:
   - `Up` and `Down` arrow keys, `Tab` or `Shift+Tab` to go up and down the list in both panes.
   - `Alt` to toggle between the right and the left panes of the switcher pop-up window.
   - `N/A` or `Backspace` to close the editor tab where the selected file is opened, or hide a tool window, if it is shown.
3. Having selected the desired file or tool window, release the `Ctrl` key. The corresponding file or tool window gets the focus and the switcher pop-up window disappears.

Tip If you want to close the switcher without selecting any file or tool window, just use the left or right arrow keys.

Navigating Between IDE Components

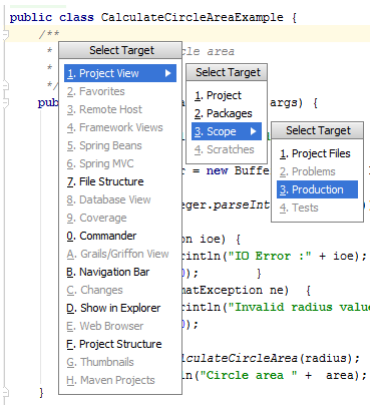
Suppose you have selected a file or member in one of the tool windows, and would like to quickly find it in the Project view. The Select Target pop-up menu moves the focus to the selected component of IntelliJ IDEA, the file system, etc.:

- Project View
- Favorites
- Remote Host
- Framework Views
- Spring Beans
- Spring MVC
- File Structure
- Database View
- Coverage
- Grails / Griffon View
- Navigation Bar
- Version Control tool window
- Show in Explorer
- Web Browser
- Project Structure
- Thumbnails
- Maven Project

To navigate to the desired component

1. On the main menu select Navigate and click Select In, or press `Alt+F1`.
   The Select Target pop-up menu shows up.

2. Use the arrow keys or the mouse pointer to select the desired component. If your target is the Project tool window, you can select the desired view: Project, Scratches etc.



Tip Double-click selected file or member in one of the tool windows, or press `Enter` to open it in the editor.

Navigating Between Methods and Tags

Since your source code can contain numerous methods, it is convenient to navigate to the beginning of the next or previous method. In the Web contents, this feature enables navigating between tags.

To navigate to the next or previous method or tag

– On the main menu, choose Navigate | Next Method / Previous Method respectively.

– Use `Alt+Up` / `Alt+Down` keyboard shortcuts.

For JavaScript code inside HTML files this behavior depends on the caret location. If the caret rests inside a JavaScript block, this means of navigation enables jumping between JavaScript functions. If the caret rests on a `<script>` tag, then navigation is performed between the tags.

To improve the visibility of your source code by having a line added automatically between adjacent methods, choose the Show Method Separators option in the Appearance page of the editor settings.

Navigating Between Test and Test Subject

On this page:

– Overview
– Jumping from a test to its test subject
– Jumping from a cass to its test

Overview

Testing support in IntelliJ IDEA provides the ability to navigate between a test and the test subject.

Jumping from a test to its test subject

1. Open the desired test class in the editor.
2. On the main menu or on the context menu of the editor, choose Navigate | Test Subject. Alternatively, press `Ctrl+Shift+T`.
   The test subject for the current test class opens in the dedicated tab of the editor and gets the focus.

Jumping from a cass to its test

1. Open the desired class in the editor.
2. On the main menu or on the context menu of the editor, choose Navigate | Test. Alternatively, press `Ctrl+Shift+T`.
   The test for the current class opens in the dedicated tab of the editor and gets the focus.

Note If a test class doesn't exist, you will be prompted to create one as described in the section Creating Test Cases Using Intention Action .

Navigating from Stacktrace to Source Code

You can easily navigate from the stack trace in the Run tool window to the source code that causes problems.

To navigate from the stack trace to a line of code

- In the Run tool window scroll to the desired stack trace line and click the link to the source file in question. The source file opens in the editor.

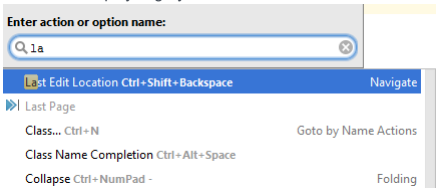
Navigating to Action

IntelliJ IDEA helps you quickly find the desired action, without digging through the menus and toolbars. The concept action covers the commands of the main menu and various context menus, commands performed through the toolbar buttons of the main toolbar and tool windows.

To find an action

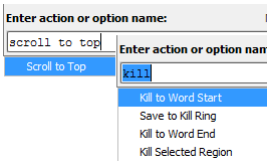
1. Choose Help | Find Action on the main menu or press `Ctrl+Shift+A`. The pop-up window that opens, shows the suggestion list of matching names. By default, this list includes the menu commands only. If you want to include the non-menu commands in the suggestion list, press `Ctrl+Shift+A` once more.

2. Start typing the desired action name. As you type, the suggestion list displays the matching names of actions. The actions that are not valid in the current context are displayed gray.



3. Double-click the desired entry in the suggestion list, or select it using the arrow keys and press `Enter`.

This way you can invoke the actions that are not mapped to keyboard shortcuts in certain schemes (for example, scroll to top, scroll to bottom, or Emacs actions, like kill rings, sticky selection, or hungry backspace).



These actions are not mapped to certain keyboard shortcuts, neither they appear in the menus. If necessary, configure keyboard shortcuts for these actions as described here .

Navigating to Braces

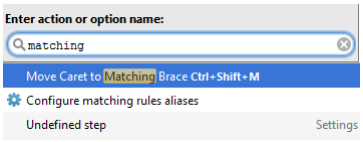
In this section:

- Navigating to the borders of a code block
- Navigating to the borders of the closest higher code block

Navigating to the borders of a code block

To navigate to the borders of a code block, do one of the following:

- To navigate to the code block start, press `Ctrl+Open Bracket`, with the caret anywhere inside the code block.
  The caret jumps to the opening brace of the current code block.
- To navigate to the code block end, press `Ctrl+Close Bracket`, with the caret anywhere inside the code block.
  The caret jumps to the closing brace of the current code block.
- To toggle between code block start or end, press `Ctrl+Shift+M`.
  You can also invoke this action (Move Caret to Matching Brace) with a Search Everywhere or Go to Action functionality:



Navigating to the borders of the closest higher code block

To navigate to the borders of the closest higher code block, do one of the following:

- To jump to the higher code block start, press `Ctrl+Open Bracket`, with the caret at the current code block opening brace .
- To jump to the higher code block end, press `Ctrl+Close Bracket`, with the caret at the current code block closing brace .

Tip Practically, you can just press `Ctrl+Open Bracket` or `Ctrl+Close Bracket` as many times as you need, until the caret is positioned at the start or end of the desired code block.

Navigating to Class, File or Symbol by Name

In this section:

- Overview
- Navigating by Name
- Tips and tricks

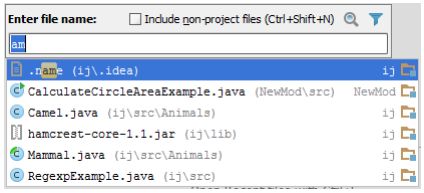
Overview

Navigate commands enable you to quickly jump to the desired classes, files, or symbols specified by names. IntelliJ IDEA suggests a look-up list of matching names, from which you can select the desired one, and open it in the editor. This navigation honors CamelCase and snake_case capitalization . Refer to the tips for detailed list of available techniques.

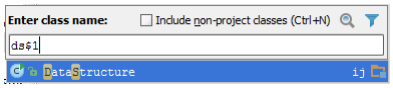
Navigating by Name

To navigate to a class, file or symbol with the specified name:

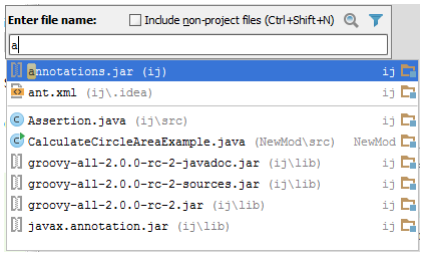
1. On the main menu, point to Navigate, and then choose Class, File, or Symbol respectively, or use the following shortcuts:
   - Class: `Ctrl+N`
   - File (directory): `Ctrl+Shift+N`
   - Symbol: `Ctrl+Shift+Alt+N`
2. In the pop-up window, start typing the desired name.
   So doing, you can enter characters located anywhere inside the desired name. As you type, the suggestion list shrinks, displaying the matching names only.
   - Class:



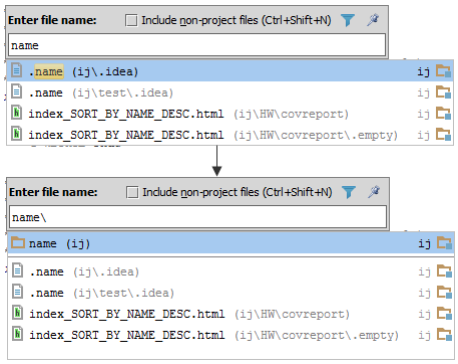
     It is also possible to navigate directly to the specified anonymous class. It is enough to specify class name and the anonymous class number, delimited with `$` character:



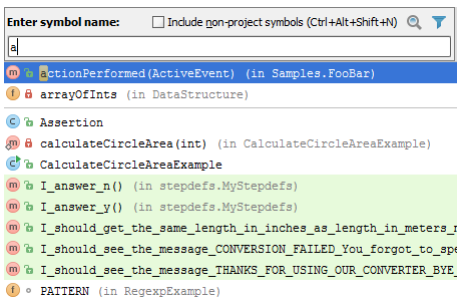
   - File:



   - Directory: use the same `Ctrl+Shift+N` shortcut as for file navigation, and type the name of the directory you are looking for, the pattern name ending with / or \:



   - Symbol:

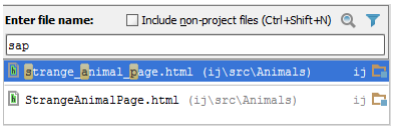


3. Double-click the desired entry in the suggestion list, or select it using the arrow keys, and press `Enter`.

Tips and tricks

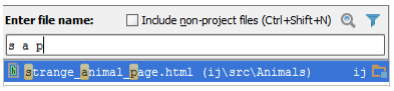
While working in the navigation pop-up window, use the following helpful techniques:

- Narrow down the search scope by selecting the file types to search in. Just click the filter, and clear the check boxes next to the file types you are not interested in.
- Include non-project files in the look-up list and thus make available matching files from SDKs and libraries.
- If the look-up list is too long, type more characters to shrink it, or click the ellipsis sign at the end of the list, to reveal its next portion.
- Type the initial letters of the CamelHumps names, for example:

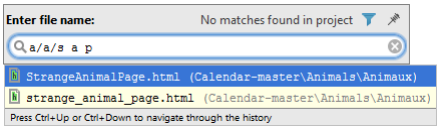


  Note that IntelliJ IDEA automatically recognizes CamelHumps and matches them to the lower case letters.

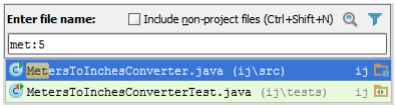
- Type any letters separated with spaces for snake_case names, for example:



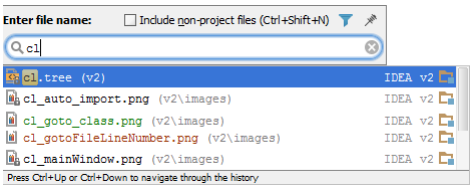
- In the navigation to file pop-up window, type letters delimited with slashes to denote nested directories:



- Type line number after a file name, delimited with a colon, to navigate to the specified line:



- Use * wildcard to represent any number of characters, though it is quite enough to type characters located in the middle of the desired name.
- If while typing in one of the Navigate to Class/File/Symbol pop-up windows you notice that you need another one, just invoke the necessary dialog box. The text you have already entered will not disappear.
- Press `Alt+F1` to invoke the Select Target pop-up window, and choose the desired IDE component.
- Note that for the projects under version control, the entries in the look-up list are color coded according to their status:

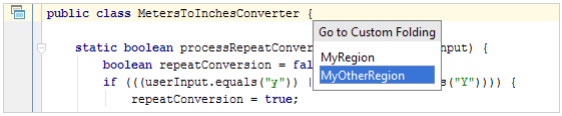


- When there is a detached editor frame with a certain file, you can opt to open this file in the main IntelliJ IDEA frame by pressing `Enter`, or activate the detached frame by pressing `Shift+Enter`.

Navigating to Custom Folding Regions

If there are code folding comments in your file (see Using code folding comments), you can navigate to corresponding folding regions like this:

1. Select Navigate | Custom Folding or press `Ctrl+Alt+Period`.
2. Select the target folding region. (The regions in the list are identified by their descriptions.)



Navigating to Declaration or Type Declaration of a Symbol

While editing your source code, you might need to navigate to the location where a particular named code reference (a symbol) has been first declared. Go To | Declaration command enables you to navigate back to the initial declaration of a symbol from any place in the source code, even if it is from inside another class, or comment.

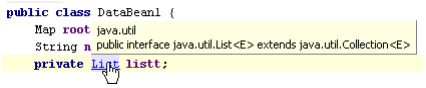
Important!

Navigate | Declaration/Type Declaration

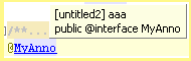
- Applies to the symbols of source code, CSS, HTML or XML tags and attributes, DTD and schema elements and attributes, and references in comments.
- Does not apply to the primitive types.

To navigate to the declaration of a symbol

1. Place the caret at the desired symbol in the editor.
2. Do one of the following:
   - On the main menu, choose Navigate | Declaration.
   - Press `Ctrl+B`.
   - Click the middle mouse button.
   - Keeping `Ctrl` pressed, point to the symbol, and click, when it turns to a hyperlink. You can also see declaration at the tooltip while keeping `Ctrl` pressed.

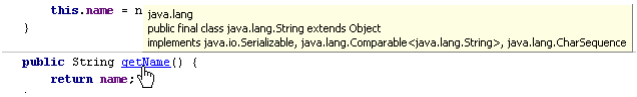


Tip Annotation types are marked in the declaration tooltips with the `@` character:



To navigate to the type declaration of a symbol

1. Place the caret at the desired symbol in the editor.
2. Do one of the following:
   - On the main menu, choose Navigate | Type Declaration.
   - Press `Ctrl+Shift+B`.
   - Press the `Ctrl+Shift` keys and hover your mouse pointer over the symbol. When the symbol turns to a hyperlink, click it without releasing `Ctrl+Shift` keys. The type declaration of the symbol opens in the editor. You can also see the declaration at the tooltip while keeping `Ctrl+Shift` pressed.



Navigating to Super Method or Implementation

IntelliJ IDEA provides an easy way to navigate up and down through the hierarchy of methods. If a method is overridden / implemented by a certain method, or overrides / implements some method itself, it is marked with an icon in the gutter area of the editor. When the mouse cursor hovers over such icon, the method information is displayed as the tooltip:

– : This method implements a method required by an implemented interface or extended abstract class.

– : This method of an interface or an abstract class is implemented by one or more descendants.

– : This method overrides a method defined by its superclass.

– : This method or field is overridden / implemented by one or more subclasses.

Use these icons, shortcuts, or menu commands to navigate to the corresponding points of origin.

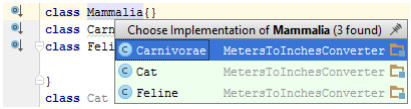
Navigating through the hierarchy of methods

To navigate up and down through the method hierarchy, do one of the following:

– Click the gutter icon and select the desired ascendant or descendant class from the list.

– On the main Navigate menu, choose Super Method, or Implementation(s) respectively.

– Press `Ctrl+U` or `Ctrl+Alt+B` for the super method or implementation respectively.



Navigating to File Path

On this page:

– Overview

– Navigating to a file path

– Viewing file path

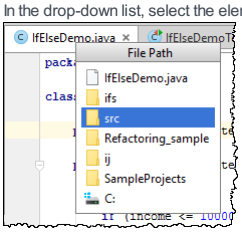
Overview

It may be helpful to open a file in a file manager (for example, in the Windows Explorer, if Windows is your OS), or the Finder. IntelliJ IDEA allows you to easily navigate to any part of a file path, right from the editor.

Navigating to a file path

To navigate to a file path from the editor

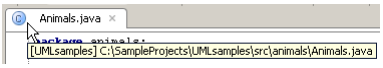
1. Hold Ctrl / ⌘ and click the relevant tab in the editor.
2. In the drop-down list, select the element you need.



The file manager shows the selected file or directory.

Viewing file path

To just view the path to a file, place the mouse pointer over the tab where the file is opened.



Navigating to Line

Navigate | Line command is the most basic way to navigate to the line with the specified number.

To navigate to a line in the editor

1. On the main menu, choose Navigate | Line, or press `Ctrl+G`.
2. In the Go To Line dialog box that appears, IntelliJ IDEA shows the current line and offset delimited by a colon. Type the target line number and, optionally, an offset, and then click OK.

The current caret position (line number and offset) is displayed in the Status Bar . You can click it to open the Go To Line dialog box.

Navigating to Next/Previous Change

If you edit a file that is under version control, IntelliJ IDEA provides several ways to move back and forth with the updates. In particular, you can use the navigation commands, keyboard shortcuts, and the change markers.

To navigate to the next/previous change in the editor, do one of the following:

- On the main menu, choose Navigate | Next / Previous Change.
- Use keyboard shortcuts `Ctrl+Shift+Alt+Down` or `Ctrl+Shift+Alt+Up`.
- Point to a change marker , and click the arrow up or arrow down buttons.

To navigate to the place of your last edit, do one of the following:

- On the main menu, choose Navigate | Last Edit Location
- Use keyboard shortcut `Ctrl+Shift+Backspace`.

Navigating to Next/Previous Error

Another method of code navigation is to move between found errors and warnings. The caret is positioned immediately before the code issue.

You can configure the way IntelliJ IDEA navigates between code issues: it can either jump between all code issues or skip minor issues and only navigate between detected errors.

In this section:

- Configuring the error navigation
- Navigating between errors or warnings

To configure the error navigation

1. Right click the Validation Side Bar .
2. On the context menu, choose one of the available navigation modes:
   - To have IntelliJ IDEA skip warnings, infos, and other minor issues, choose Go to high priority problems only.
   - To have IntelliJ IDEA jump between all detected code issues, choose Go to next problem.

To navigate between errors or warnings, do one of the following

- On the main menu, choose Navigate | Next / Previous Highlighted Error.
- Use keyboard shortcuts `F2` and `Shift+F2` respectively.

Navigating to Recent

In this section:

- Navigating to a recently opened file
- Navigating to a recently edited file
- Navigating to the latest edit location
- Navigating to the next edit location
- Using multi-selection in the lists of recent files

To navigate to a recently opened file

1. On the main menu, choose View | Recent Files or press `Ctrl+E`.
2. From the Recent Files pop-up window that opens select the desired file.

To navigate to a recently edited file

1. On the main menu, choose View | Recently Changed Files or press `Ctrl+Shift+E`.
2. From the Recently Edited Files pop-up window that opens select the desired file.

Tip The recently opened or recently modified files are selected from the history list. The number of entries in the history list is configurable in the Recent file limit field in the Editor settings page.

- Navigating to recent files applies to the search results as well. By pressing `Ctrl+E` in the Find tool window, you can have the list of recent search results shown.

To jump to the latest edit location

- Do one of the following:
  - On the main menu, choose Navigate | Last Edit Location.
  - Press `Ctrl+Shift+Backspace`.

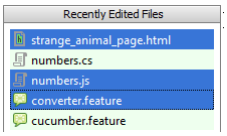
To jump to the next edit location

- On the main menu, choose Navigate | Next Edit Location.

This action is not bound to any shortcut. An interested user can do it as described in the section Configuring Keyboard Shortcuts .

Using multi-selection in the lists of recent files

- To select non-adjacent files, use `Ctrl` + mouse click.
- To select adjacent files, use `Shift` + mouse click.



Navigating to Navigated Items

You can go back and forth along the items that have already been navigated to. This feature applies to the items reached using all navigation commands except for the simplest ones, such as arrow keys, Pageup, Page Down, Home and End.

To navigate to the navigated items

Do one of the following:

- On the main menu, choose Navigate | Back / Forward.
- Use keyboard shortcuts `Ctrl+Alt+Left`, or `Ctrl+Alt+Right`.
- On the main toolbar, click or .

Note On an OS X computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.

Navigating with Breadcrumbs

On this page:

– Overview

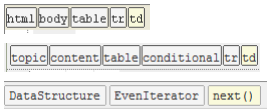
– Enabling breadcrumbs

– Navigating between a breadcrumb and the source code

Overview

In the HTML and XML context and in Java or PHP classes, you can navigate through the source code using breadcrumbs that are displayed above the main editor window.

The breadcrumbs show the elements of a XML/HTML file, Java or PHP class opened in the active editor tab :



To have breadcrumbs shown, you have to enable them.

Enabling breadcrumbs

To have breadcrumbs displayed

1. Open the IntelliJ IDEA Settings/Preferences dialog.
2. Under the Editor node, click General | Appearance.
3. In the Appearance page, select the Show breadcrumbs check box.

Navigating between a breadcrumb and the source code

To navigate with breadcrumbs

– Click the breadcrumb with the relevant tag, or the Java or PHP class element. The cursor jumps to the corresponding element in the source code.

Navigating with Favorites Tool Window

The Favorites tool window consolidates the views of the project's favorite items, bookmarks and breakpoints. Choosing the appropriate viewing mode for this tool window helps you have all information at hand, and navigate through the items with ease.

To jump from an item in the Favorites tool window to the file in the editor, do one of the following

- Double-click the selected item.
- On the context menu of the selected item, choose Jump to Source.
- Press F4.

The corresponding file opens in the editor. In case of a bookmark or a breakpoint, the caret rests at the beginning of the line with this bookmark or breakpoint.

Navigating with Navigation Bar

Use the Navigation Bar as a handy tool to find your way across the project.

To navigate to a file using the Navigation bar

1. Press `Alt+Home` to activate the Navigation bar.
2. Use the arrow keys or the mouse pointer to locate the desired file.
3. Double-click the selected file, or press `Enter` to open it in the editor.

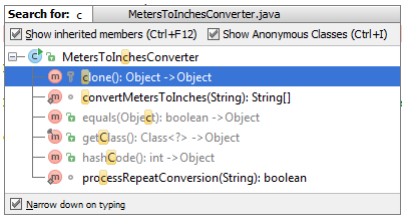
Navigating with Structure Views

Use the Structure pop-up window or the Structure tool window to quickly jump to the desired member of a file in the editor. The Structure views provide quick navigation for all supported file types.

Besides that, navigation with the Structure pop-up window is also available for diagrams .

To navigate to a member in the editor

1. Choose Navigate | File Structure on the main menu or press `Ctrl+F12`.
2. In the File Structure pop-up window that opens select the Narrow down the list on typing check box and start typing the desired member name.
   You can include the members, inherited from the parent classes, by checking the option Show inherited members or pressing `Ctrl+F12` again.



3. Use the navigation keys to select the desired node. Then do one of the following:
   - If the cursor rests on a top or intermediate node (for example, class or element), double-click this node or press `Enter` to expand it in the Structure pop-up, or press `F4` to jump to its declaration in the editor.
   - If the cursor rests on a leaf node (for example, a member , or lowest-level element), double-click this node or press `Enter` to jump to its declaration in the editor.

   In the case of an inherited member, the respective parent class opens in the editor.

You can also use the Structure tool window (`Alt+7`). This view is flexibly configurable and useful for many tasks, apart from navigation. However, the File Structure pop-up window is the easiest way for quick navigation.

Searching Through the Source Code

IntelliJ IDEA provides extensive search and replace capabilities, which includes basic search and replace, search and replace in paths, finding usages, code-aware structural search and more. Some of the replacement facilities (like renaming classes and members) are performed by means of refactorings .

All the search commands can be found under the Find node of the Edit menu.

In this part you will learn how to:

- Find and replace text in the current file.
- Find specific word .
- Search and replace across the project .
- Find usages .
- Highlight usages .
- View usages of a symbol across the project, and jump to the desired usage .
- Work with the search results .
- Perform structural search and replacement using a template.

Finding and Replacing Text in File

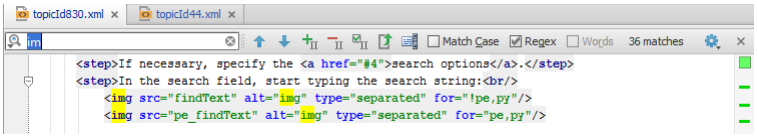
The standard facility helps you find and replace text strings in the active editor.

On this page:

- Search through the current file
- Replace in the current file
- Working with search results
- Search and replace options

Search through the current file

1. From the main menu, choose Edit | Find | Find, or press `Ctrl+F`. The search pane appears on top of the active editor.
2. If necessary, specify the search options .
3. In the search field, start typing the search string:



As you type, the first occurrence of the search string after the current cursor position is selected; the other occurrences are highlighted in the editor. In addition, the matching occurrences are marked in the right gutter with stripes.

```
race = function() {
    var runners, winner;
    winner = arguments[0], runners = 2 <= arguments.length ? __slice.call(arguments, 1)
    return print(winner, runners);
};
```

4. Explore the search results .

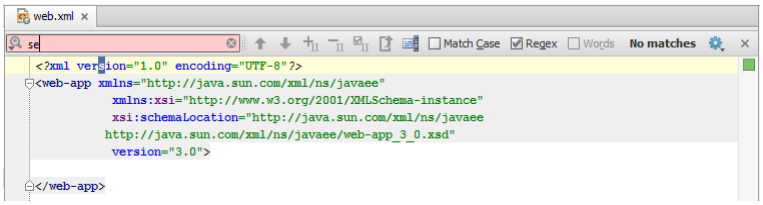
Replace in the current file

1. From the main menu, choose Edit | Find | Replace, or press `Ctrl+R`. The search and replace pane appears on top of the active editor.
2. If necessary, specify the search and replace options .
3. In the search field, start typing the search string. As you type, the matching occurrences are highlighted in the editor, and a Replace pop-up dialog box opens at the first occurrence, suggesting you to replace the current occurrence, or all of them, with an empty string.
4. Start typing the replacing string.
5. Explore the search results , and, using the buttons of the replace dialog box, replace occurrences as required. See Search and replace options below.

Working with search results

- To initiate a new search, do one of the following (depending on the current focus):
  - If the editor has the focus, press `Ctrl+F`.
  - If the search field has the focus, press `Ctrl+A`

  In both cases, the existing search string will be selected, and you can start typing a new one.
- To jump between occurrences, do one of the following:
  - Press `Shift+F3` (jump to previous selection) or `F3` (jump to next selection).
  - Use the ↑ or ↓ buttons in the Search pane.
  - Click the gutter stripes.
- The search pane shows the number of found occurrences. If no matches are found, the search pane is highlighted with red:



- Use Code completion in the Find and Replace panes. Start typing a search string, press `Ctrl+Space`, and select the appropriate word from the suggestion list.
- Use the recent search history: with the search pane already open, click to show the list of recent entries.
- With the Find and Replace pane already opened, use `Ctrl+R` or `Ctrl+F` to toggle between panes. Thus, the search and replace strings are preserved.
- To cancel the operation and close the pane, press `Escape`.
- Use multiple selection (multiselection). For example, if a certain string has been highlighted as a search result, it is possible to add an occurrence of this string to multiple selection by clicking (`Alt+J`), delete an occurrence from multiple selection using (`Shift+Alt+J`), or add all found occurrences to multiple selection using (`Ctrl+Shift+Alt+J`).

Search and replace options

Item	Description	Search/Replace
	Click this button to show the history of the recent entries.	Search, replace
	Click this button to clear the search field.	Search, replace
↑ ↓	Click these buttons to navigate through the occurrences of the search string.	Search, replace

	Click this button to add the next found occurrence to a multiple selection.	Search
	Click this button to remove the found occurrence from a multiple selection.	Search
	Click this button to create a selection that contains all the found occurrences.	Search
	Click this button to show search results in the Find tool window .	Search, replace
Match Case	If this check box is selected, IntelliJ IDEA will distinguish between upper and lowercase letters while searching.	Search, replace
Regex	If this check box is selected, the search string will be perceived as a regular expression , and the replacement preview is shown in a tooltip (Refer to the section Example).	Search, replace
Words	if this check box is selected, IntelliJ IDEA will search for whole words only, that is, for character strings separated with spaces, tabs, punctuation, or special characters. This check box is disabled, if the Regular expressions check box is selected.	Search, replace
Preserve Case	If this check box is selected, IntelliJ IDEA retains the case of the first letter and the case of the initial string in general. For example, *MyTest* will be replaced with *Yourtest* if you specify *yourtest* as the replacement. This check box is disabled, if Regular expressions check box is selected.	Replace
In Selection	If this check box is selected, search and replacement will be confined to the selected text only.	Replace
Replace	Click this button to replace the current occurrence and proceed to the next one.	Replace
Replace all	Click this button the replace all found occurrences in the current file, or in the selection.	Replace
Exclude/Include	Click Exclude button to skip the current occurrence and exclude it from the Replace all operation. The button for this occurrence changes to Include.	Replace
	Click this button to invoke the list of additional options. Checking the corresponding option confines the search to the specified scope, while the other occurrences are ignored.	Search, replace

Tutorial: Finding and Replacing Text in File Using Regular Expressions

On this page:

– Example code
– Finding and replacing a string using regular expression
– Changing case of the characters

Example code

Consider the following XML code fragment:

```
<new product="ij"
                     category="105" title="Multiline search and replace in the current
                     file" />
                     <new product="ij" category="105" title="Improved search and replace in the current
                     file" />
                     <new product="ij" category="105" title="Regexp shows replacement preview"
                     />
                     ...
```

Finding and replacing a string using regular expression

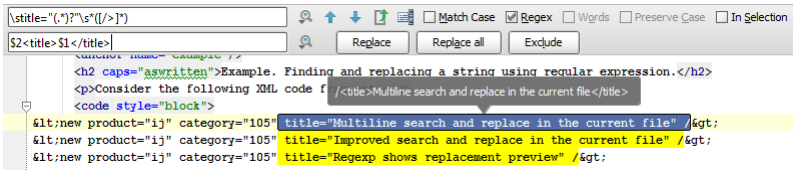
Suppose you want to replace an attribute within an element (`title`) with an expanded tag `<title></title>` , which contains some arbitrary string in double-quotes within.

This is how it's done.

1. With the XML file in question opened in the editor, press `Ctrl+R` . The Replace pane appears on top of the editor.
2. Since you want to replace all the `title` attributes, regardless of the actual strings contained therein, use regular expressions. Make sure that the check box Regex is selected. Thus everything you type in the Search and Replace fields will be perceived as the regular expressions.
3. In the Search field, start typing regular expression that describes all `title` attributes. Note that though the regular expression `\stitle=".*?"\s*[/>]*` matches contents of the `title` attribute, it is recommended to capture the groups for referencing them in the Replace field:

   ```
   \stitle="(.*)?"\s*(/>*)
   ```

   Note that for the regular expressions replacement preview is shown at the tooltip:

4. Then, in the Replace field, type the following regular expression:

   ```
   $2<title>$1</title>
   ```

5. Click Replace, or Replace All.

As you see, the second capture group (`/>`) is moved ahead to close the <new> element, while the first capture group `<` , which matches any string in double quotes, is moved to the element `<title>` .

Changing case of the characters

Suppose now that you want to change characters os the search strings. Again make sure that Regex check box is selected.

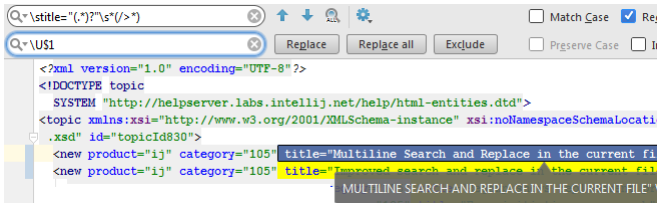
In the Search field, type the search expression:

```
\stitle="(.*)?"\s*(/>*)
```

Next, fill in the Replace field with the following expression:

```
\U$1
```

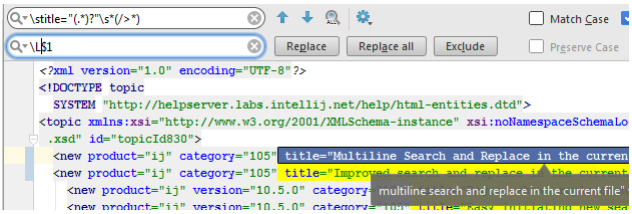
The found occurrences are replaced with the upper-case characters:



Next, let's make the strings all lower-case. To replace occurrences with the lower-case characters, type the following replacement string:

```
\L$1
```

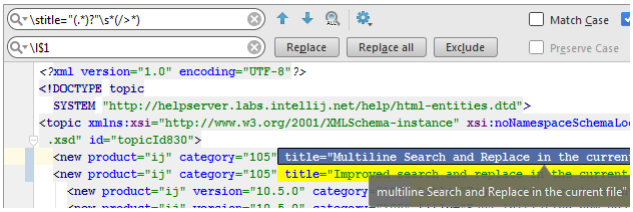
Then the suggested replacement will be:



And finally, if you want change the case of the first letter only, type the following replacement string:

```
\l$1
```

IntelliJ IDEA suggests the following replacement:



Finding and Replacing Text in Project

IntelliJ IDEA extends search and replace capability to the entire project, specific module, or any directory with its nested hierarchy. Explore search results in the Find tool window .

On this page:

- Finding a piece of text in all the files within the specified path
- Replacing a piece of text in all the files within the specified path
- Toggling between the Find in Path and Replace in Path dialog boxes

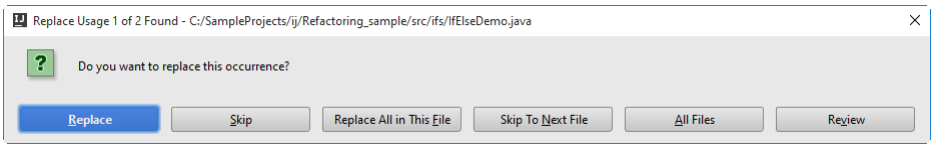
Finding a piece of text in all the files within the specified path

1. On the main menu, choose Edit | Find | Find in Path, or press `Ctrl+Shift+F` .
2. In the Find In Path dialog box, specify the following options:
   - The text to find. Type the text explicitly, or specify a pattern using a regular expression, or select a previously used piece of text or a pattern from the recent history drop-down list.
     If you specify the search pattern through a regular expression, use the `$n` format in back references (to refer to a previously found and saved pattern).
   - Search scope (project , module or directory).
   - Search options (case sensitivity, whole words, and regular expressions).
3. Click Find.

If the search takes too long time, click Background in the search progress window. In this case the search progress is indicated in the Status bar.

Replacing a piece of text in all the files within the specified path

1. Do one of the following:
   - On the main menu, choose Edit | Find | Replace in Path.
   - Press `Ctrl+Shift+R`
   - Being in the Find In Path dialog box, press `Ctrl+Shift+R` to switch to Replace In Path dialog box.
2. In the Replace In Path dialog box, specify the search and replace strings, the search options, and the scope. Type the search and replacement text explicitly, or specify patterns using regular expression, or select a previously used piece of text or a pattern from the recent history drop-down list.
   - If you specify the search and/or replacement text through a regular expression, use the `$n` format in back references (to refer to a previously found and saved pattern).
   - To use a backslash character `\` in a regular expression, escape the meaningful backslash by inserting three extra backslashes in preposition: `\\\\` .
3. Click Find. IntelliJ IDEA displays the encountered occurrences of the search string in the Find tool window , selects the first occurrence and opens the file with this occurrence in the editor and moves the focus to it.
   At the same time, IntelliJ IDEA opens the Replace Usage dialog box, with the full path to the encountered occurrence in the title bar:



Do one of the following:

- To have the selected occurrence replaced, click Replace.
- To preserve the selected occurrence and move to the next one, click Skip.
- To have all the occurrences of the search string in the currently active tab replaced, click Replace All in This File.
- To preserve the occurrences of the search string in the currently active tab (any) and move to the next file, click Skip to Next File.
- To have all the detected occurrences replaced, click All Files.
- To switch to the manual mode, click Preview. The Replace Usage dialog box closes and the focus moves to the Find tool window. Do one of the following:
  - Browse through the list of detected occurrences, select the ones you want to replace and then click Replace Selected.
  - To have all the occurrences changed click Replace All.

Tip To learn about changing case of the literals, refer to the example , and to Regular Expression Syntax Reference .

Toggling between the Find in Path and Replace in Path dialog boxes

- To switch from the Find In Path to Replace In Path dialog box, press `Ctrl+Shift+R` . To switch from the Replace In Path to Find In Path dialog box, press `Ctrl+Shift+F` .

Finding Word at Caret

Search for a word at caret enables you to quickly find the exact match for the current word, without changing any search options. With the match once found, you can navigate between the occurrences of the term.

To find the word at caret, do one of the following

- On the main menu, choose Edit | Find | Find Word At Caret.
- Use `Ctrl+F3` keyboard shortcut.

To navigate between the occurrences of the word at caret

1. Press `F3` to go to the next occurrence.
2. Press `Shift+F3` to go to the previous occurrence.

Finding Usages

Search for usages is an important part of the code analysis, which enables you to clarify dependencies. IntelliJ IDEA suggests several types of search for usages:

- Finding Usages in Project
- Finding Usages in the Current File
- Highlighting Usages
- Viewing Usages of a Symbol
- Viewing Recent Find Usages

Finding Usages in Project

IntelliJ IDEA provides different search options depending whether you are searching for usages of a class, method, field, parameter, or throw statements, and extends search for usages to the files in supported languages. For example, in CSS, XML and HTML files you can search for the usages of styles, classes, tags and attributes.

Search for usages extends to the Cucumber step definitions as well.

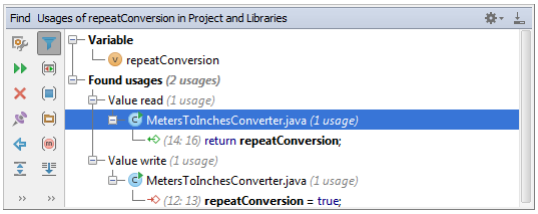
Explore search results in the Find tool window .

To find usages of a symbol in a project

1. Select a symbol to find usages for. To do that, place the caret within the desired symbol in the editor, or click the symbol in the Project tool window . You can also select symbol in the UML Class diagram

2. Do one of the following:
   - On the main menu, choose Edit | Find | Find Usages
   - Choose Find Usages on the context menu
   - Press `Alt+F7` .

3. In the Find tool window , explore search results. Use the button to represent search results in meaningful groups by type of usage.



While analyzing the search results, you can at any time open the search options dialog box by clicking click in the Find tool window or by pressing `Ctrl+Shift+Alt+F7` .

To find usages of a symbol in the current file

1. Click the desired symbol in the editor, or in the Structure view.
2. On the main menu, choose Edit | Find | Find Usages in File, or press `Ctrl+F7`. The encountered usage is highlighted in the editor.

For a field, you can specify whether you want to search for its accessor methods in the Search Accessors dialog box.

Highlighting Usages

On this page:

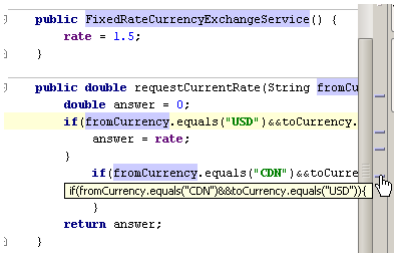
– Introduction
– Configuring automatic highlighting
– Highlighting usages in the current file
– Navigating among usages
– Removing highlighting

Introduction

The search command Highlight Usages in File (`Ctrl+Shift+F7`) makes it possible to visualize usages of a symbol in the current file.

In particular, it is possible to highlight implemented methods of interfaces.

All found usages of a symbol in the current file are highlighted and color-coded, as defined in the Colors and Fonts settings page, to represent read or write access to the symbol. In addition to the highlights of occurrences in text, the stripes of the same colors appear in the marker bar, accompanied with tooltips.



The behavior of usage highlighting is configurable: you can make IntelliJ IDEA show usages of a symbol at caret automatically, or invoke it with a command.

Configuring automatic highlighting

To configure highlight usages behavior

1. Open the Settings/Preferences dialog .
2. In the Editor page of the IDE Settings, select the check box Highlight usages of element at caret to enable automatic usage highlighting.

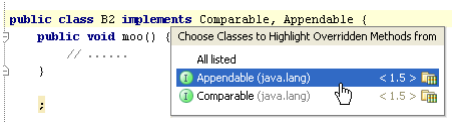
Highlighting usages in the current file

To highlight usages of a symbol in the current file

1. Place the caret at the selected symbol in the editor. If automatic usages highlighting is enabled, see all its occurrences in the current file highlighted. Otherwise, proceed to the next step.
2. On the main menu, choose Edit | Find | Highlight Usages in File, or press `Ctrl+Shift+F7` .

To highlight overridden methods

1. In a class that implements one or more interfaces, place the caret at the `implements` keyword in the class declaration.
2. On the main menu, choose Edit | Find | Highlight Usages in File, or press `Ctrl+Shift+F7` . A list of implemented interfaces shows up:



3. Select the interface, whose methods you want to highlight, and press `Enter` .

Navigating among usages

To navigate among usages, do one of the following

– Click on a stripe in the marker bar to navigate to the respective usage location.

– Use `F3` and `Shift+F3` keyboard shortcuts, to navigate to the next and previous usages respectively.

Removing highlighting

To remove highlighting of usages

– Press `Escape` .

Viewing Usages of a Symbol

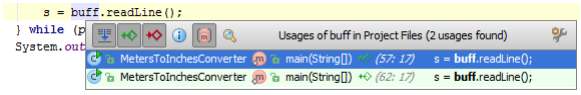
Using the Show Usages function, you can bring up a list of the usages of a symbol across the whole project. So doing, the pop-up window with the list of usages of a symbol features a toolbar with the following buttons:

Icon	Shortcut	Description
	Ctrl+F	Merge usages of the symbol from the same line.
	Ctrl+R	Show read access to the symbol.
	Ctrl+W	Show write access to the symbol.
	Ctrl+I	Show usages in the import statements.
	Ctrl+M	If this button is pressed, the found usages show under the corresponding method nodes.
	Alt+F7	Show search results in the Find Usages tool window.
	Ctrl+Shift+Alt+F7	Open the Find Usages dialog box for the selected symbol where you can change the search options.

Tip In addition to the ability of viewing usages, you can use this function as a quick means of navigation.

To view the usages of a symbol across the project

1. Place the caret at the desired symbol in the editor.
2. On the main menu, choose Edit | Find | Show Usages, or press Ctrl+Alt+F7.
3. Examine and analyze the detected occurrences of the symbol:
   - Use the toolbar buttons to present search results in the desired way.
   - To jump from search results to a line of source code, click the desired entry.
   - To close the list, press Escape.



4. If necessary, customize the search options in the search options dialog box. To invoke the dialog box, do one of the following:
   - In the Show Usages pop-up window, click .
   - Press Ctrl+Shift+Alt+F7.

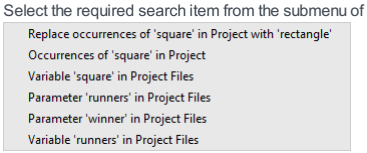
Viewing Recent Find Usages

With IntelliJ IDEA you can easily navigate to the recent search results if any find usages action took place during the IntelliJ IDEA session.

To view recent find usages

1. Select Edit | Find | Recent Find Usages on the main menu.

2. Select the required search item from the submenu of the recent search results:



The selected search shows in the Find tool window.

Structural Search and Replace

Structural Search and Replace (SSR) performs search and replace in the supported languages code across the whole project, taking advantage of the IntelliJ IDEA's awareness of the syntax and code structure of the supported languages.

IntelliJ IDEA finds, and if required, replaces fragments of source code, using the search templates .

Structural Search and Replace is helpful when you need to browse through or modify an extensive code base, find changes in libraries, explore the source code for specific constructs, or refactor the source code.

Tip To see the list of supported languages, open the dialog box Structural Search and Replace Dialogs and click the drop-down list File type.

In this section:

- Search Templates
- Structural Search and Replace - General Procedure
- Creating and Editing Search Templates
- Structural Search and Replace Examples

Search Templates

Search templates are the essential part of Structural Search and Replace feature. Like Live templates , the search templates consist of plain text and one or more template variables.

A valid search or replacement template represents one of the following Java constructs:

- Expression, for example `new SomeExpression()`
- Statement, or sequence of statements, for example `q = 1;`
- Class designator, for example `class Booking implements Serializable`
- Line or block comments, for example `/** Created in IntelliJ IDEA */`.
- `@Modifier` annotations.

IntelliJ IDEA provides a collection of predefined search templates, that match the various statements, expressions, classes and their members, XML and HTML constructs, and more. You can use these templates for structural search and replace, and also as a basis for creating your own search templates.

The search templates make use of the variables, which are the strings surrounded with `$` characters, for example `$expression$`. Symbols in the source code, `String` literals, and comments can be referred to by means of variables.

Variables in a template are subjected to certain constraints that help you refine your search and limit it to the desired matches:

- Text constraints are text patterns to match against. These constraints can be plain text, or regular expressions, and can contain references to symbols.
- Number of occurrences defines how many sequential elements (in a parameter, declaration or statement list) a variable can include and whether a variable is required to be present in a pattern or not. If the number of occurrences is 1, only one symbol can match the variable. If the number of occurrences is null, it means that an element could be missing.
- Expression constraints apply semantic conditions to the search, for example locate the symbols that are read or written to.
- Script constraints are used when items to search for are more than a plain match. If you are looking for certain language constructs (for example, constructors with the specified number of parameters, or members with the specified visibility modifiers), apply constraints described as Groovy scripts.

In search templates, the following simplifications can be used:

- Method body can be omitted.
- If no access modifier is indicated, any access modifier will be honored.
- Short class names (instead of fully qualified names) are used in the templates and constraint fields.
- Using `class $Class$` as a template, results in finding anonymous classes as well.
- Templates for comments and documentation comments should contain variables and constructs with correct comment and JavaDoc syntax.

Structural Search and Replace - General Procedure

This section outlines the general SSR procedure. Refer to the section Structural Search and Replace Examples for typical use cases.

To find and replace source code structurally, follow these general steps:

1. On the main menu, choose Edit | Find | Search Structurally, or Edit | Find | Replace Structurally.
2. In the dialog box that opens, define the search template . In brief, defining a search template involves the following steps:
   - Type the desired construct in the Search template text area, or use one of the pre-defined search templates by clicking the Copy existing template button.
   - Specify the constraints to be imposed on the variables within the search template. To do that, click the Edit variables button. All the variables contained in the search template are listed in the Variables pane of the Edit Variable dialog box.

   Refer to the section Creating and Editing Search Templates for the detailed description of procedure.
3. In case of the structural replace, specify the replacement pattern, and define variable constraints as required.
4. Specify the search and replace options, in particular, the number of occurrences to be matched, and the type of files to be analyzed.
5. Specify the scope to perform the structural search and replace in. To do that, click the down arrow in the Scope list, and select one of the pre-defined scopes, or click the ellipsis button, and configure the desired scope in the Scopes dialog box.
6. Click Find. The detected occurrences are displayed in the Find tool window .
   Please note that in case of replacement, you can select the desired matches in the search results and click the Preview Replacement button. The corresponding occurrence is highlighted in the source code.

Creating and Editing Search Templates

You can create search and replace templates from scratch, just typing the code in the text area of the Structural Search / Replace dialog box. However, there is a collection of the predefined search templates that you can use as prototypes for your own templates. All custom templates appear in the list of existing search templates, under the node User defined.

To create a search template, follow these general steps:

- On the main menu, choose Edit | Find | Search Structurally.
- Do one of the following:
  - Type the code of your template in the Search template text area.
  - Click the Copy existing template button, and in the Existing Templates dialog box, select the desired template as a prototype. The source code of the selected template appears in the Search template text area, where you can change it as required.
- If you need to configure the template variables, click the Edit variables button. Edit Variables dialog box appears.
  In the Variables column, select a variable you want to configure, and specify the constraints that will apply to this variable. See the dialog reference page for the detailed description of constraints.

  Repeat the process for the other variables, as required, apply changes and close the dialog box.
- Click the Save Template button.
- In the Save Template dialog, type the name of the new template, and click OK.

Tip You can create a new template in the Existing Templates dialog. To do that, click the + button on the toolbar. This opens the Structural Search dialog, with the empty template field. To define the custom template, follow the steps described above.

Structural Search and Replace Examples

This section provides a collection of structural search and replace examples:

- Finding one statement
- Finding a method call
- Searching in comments and strings
- Searching for constructors of the class
- Adding try/catch/finally code
- Finding all descendants of a class, or all classes implementing a certain interface
- Using @Modifier for finding package local and instance methods
- Using "Contained in Constraints" field in a search
- Searching for XML and HTML tags, attributes and their values
- Using script constraints

One statement

```
$Statement$;
```

Increasing the number of occurrences count to a certain number, you can find sequences of statements that contain up to the specified number of elements.

Method call

```
$Instance$.$MethodCall$($Arguments$)
```

This template matches method call expressions. If the number of occurrences is zero, it means that a method call can be omitted.

If statement

```
if ($Expr$)
                    {
                    $ThenStatements$;
                    }
                    else {
                    $ElseStatements$;
                    }
```

Search in comments and/or string literals

Consider one wants to find comments or literal containing 'foo'. Search template would be like `$SomethingWeWantToFind$` or `"$SomethingWeWantToFind$"`. In case one wants to find comments/string containing some particular words (say, foo as a word), this should be specified as a text constraint.

Search for constructors of the class

The search template Constructors of the class with default variable settings lets you find only one constructor in each class within the specified scope. If the class has several constructors then to find more than one, you need to set the Maximum count option of Occurrences count to Unlimited in the `$Class$` variable. For more information, see Edit Variable Dialog .

Note that the class declarations will be also included into the find occurrences' list.

Add try/catch/finally code

If one wants to replace a statement with a `try/catch/finally` construct, the following pair of search and replace templates can be suggested. The search template is:

```
$Statements$;
```

with a certain maximum number of occurrences specified as a constraint.

The replacement template is:

```
try {
                    $Statements$;
                    }
                    catch(Exception ex)
                    {
                    }
```

Finding all descendants of a class or all classes that implement a certain interface

Consider the following search templates:

```
class $Clazz$ extends
                    $AnotherClass$
                    {}
```

or

```
class $Clazz$ implements
                    $SomeInterface$
                    {}
```

As the text constraint for the variables $AnotherClass$ or $SomeInterface$, specify the name of the base class or implemented interface.

Finding all such methods

To look for the different implementations of the same interface method, use the following search template:

```
class $a$ {
                    public void $show$();
                    }
```

Specify text constraint for the `$show$` variable, and enable the option This variable is the target of the search.

Using @Modifier for finding package local and instance methods

IntelliJ IDEA suggests pre-defined templates for the package local and instance fields of a class. These templates make use of the @Modifier annotation, which helps describe search target, when there is no way to express it using the natural language means.

However, if you need to search for package local or instance methods, you will have to create the corresponding search templates yourself, applying the @Modifier annotation.

To specify criteria for finding all methods with the visibility modifiers package local and instance, use the following search template:

```
class
                    $Class$ {
                    @Modifier("packageLocal")
                    @Modifier("Instance" )
                    $ReturnType$ $MethodName$($ParameterType$
                    $Parameter$);
                    }
                    }
```

Using "Contained in Constraints" field in a search

The existing example uses the following template:

```
LOG.debug($params$);
```

Placing `if('_a) { '_st*; }` where `_a` and `_st` are variables and `*` denotes zero or more occurrences in Contained in Constraints field and selecting Invert condition check box of Complete Match variable will result a search of logging statements that are not contained in the `if` statement.

Searching for XML and HTML tags, attributes, and their values

The simplest template that is used to search for a tag is:

```
<$a$/>
```

By placing constraints on the variable `$a$`, you can specify which tags you want to find. For example, if you specify the text/regexp constraint `app.+`, you'll find the tags whose names start with `app`.

A more versatile template for searching in XML and HTML is:

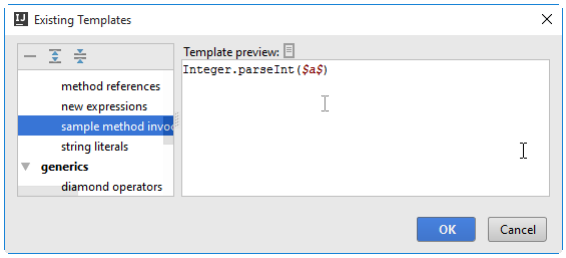
```
<$tag$ $attribute$="$value$"/>
```

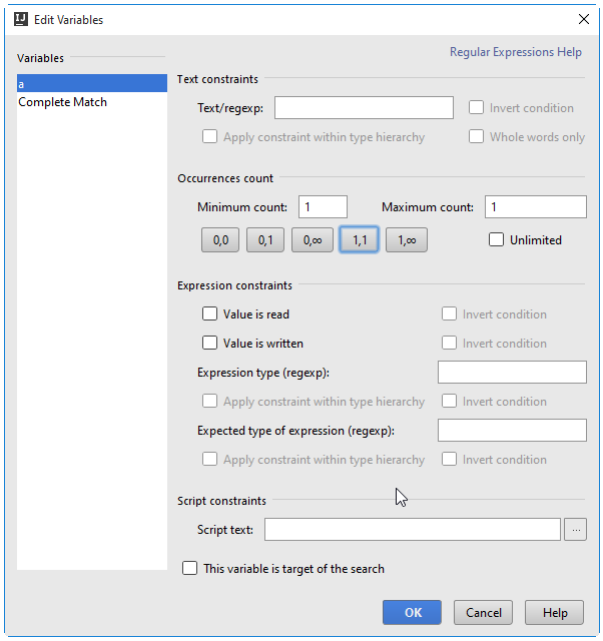
By using this template with properly specified search settings and constraints, you can find practically anything that may occur in XML or HTML. For example, if you specify the text/regexp constraint `width` for the variable `$attribute$`, you'll find all the tags that have the `width` attribute.

Using script constraints

IntelliJ IDEA structural search lets you use advanced constraints that cannot be specified using UI.

See the following template as an example:





The constraint is specified using Groovy scripting language and IntelliJ IDEA PSI API for the language you are searching. The Groovy constraint is applicable to any language to which the structural search can be applied.

Working With Search Results

The search results display in the Find tool window . The results of each search display in a separate tab.

Using the controls and the context menu commands of this tool window as well as the main menu, you can:

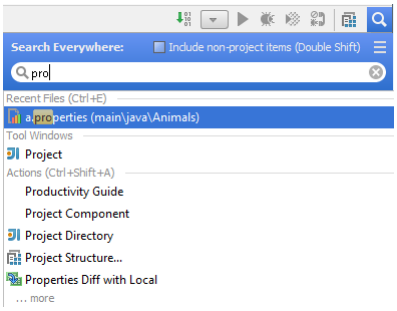
- Navigate to source code .
- Exclude and include search results in refactoring.
- Add selected search results to favorites .
- Show the results of the recent find usages.
- Perform version control operations using the specific VCS, associated with the parent directory of the usage.
- Hide excluded search results from showing them in the Find tool window (`Alt+Delete`) .

Searching Everywhere

IntelliJ IDEA makes it possible to look for any item of the source code, databases, actions, elements of the user interface, etc. in a single action.

To search everywhere, follow these steps

1. Do one of the following:
   - Click in the upper-right corner of the IntelliJ IDEA window.
   - Double-press `Shift`
2. In the pop-up window that opens, start typing the search string, to narrow down the suggestion list:

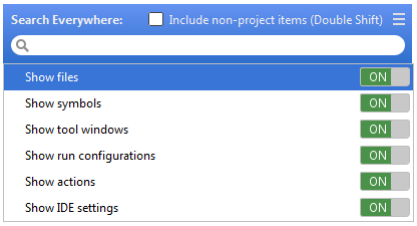


   The search string can be a name of a symbol or action, an option, or an abbreviation of an action.
3. Select the desired target from the suggestion list. The further behavior depends on the search target. IntelliJ IDEA may silently navigate to the desired source code, show the desired setting, or a pop-up list of Find Action.
   You can also just press the desired shortcut, to switch from the Search everywhere pop-up window to any other navigation and search action: for example, press `Ctrl+N` to jump to the selected class, or `Ctrl+Alt+S` to show the corresponding option in the Settings dialog.

Tip IntelliJ IDEA allows viewing the history of the previous searches with the help of the left arrow key. Search Everywhere remembers last 50 queries finalized with `Enter` key.

To configure search everywhere scopes

1. In the Search Everywhere dialog, click .
2. Turn on or off the desired search scopes:



Compiling Applications

This section describes how to:

- Configure compiler settings .
- Compile source files, modules or packages .
- Make a module .
- Make or rebuild a whole project.
- Work with the results of compilation .
- Package a Java module into a JAR file .

Note that the way the module dependencies are ordered may be very important for the compilation to succeed. See Order of dependencies.

Compiler and Builder

The Java builder of IntelliJ IDEA builds, or brings together source files, external libraries, properties files and configurations to produce a living application. The builder uses a compiler that works according to the Java specification.

The compiler treats encountered problems as errors and warnings. Unlike errors, the warnings do not prevent successful compilation. Both warnings and errors are reported in the Messages window.

The following key principles underlie the notion of compilation:

- All compilation tasks are performed in a separate process, fully independent from the main IDE's process, to get rid of the "out of memory" and other resources contention.
- Java compiler is used "in-process" via Java API for better performance and more flexible annotation processors management.
- Compilation is automatically triggered by events from the file system of the IDE, if such option is enabled .

To build your project, you do not have to leave your editing environment: with IntelliJ IDEA you can compile, build and run your source code straight away.

Note that the way the module dependencies are ordered may be very important for the compilation to succeed. See Order of dependencies.

In this part:

- Compilation Types
- Supported Compilers
- Build Process

Compilation Types

IntelliJ IDEA suggests several ways of compiling and building applications. The corresponding commands are available in the Build menu.

- Compile <compilation_scope>. All the source files in the specified scope are compiled. The scope in this case may be a file, a package, etc.

  Refer to the section Compiling Target for details.
- Make Project. All the source files in the entire project that have been modified since the last compilation are compiled. Dependent source files, if appropriate, are also compiled. Additionally, the tasks tied to the compilation or make process on modified sources are performed. For example, EJB validation is performed if the corresponding option is enabled on the Validation page .

  Refer to the section Making Project for details.
- Make Module. Compiled are all the source files that have been modified since the last compilation in the selected module as well as in all the modules it depends on recursively.

  Refer to the section Making Module for details.
- Rebuild Project. All the source files in the project are recompiled. This may be necessary when the classpath entries have changed, for example, SDKs or libraries being used added, removed or altered.

  Refer to the section Rebuilding Project for details.

Supported Compilers

Currently IntelliJ IDEA supports the following Java compilers:

- Javac. This compiler is taken from the Java SDK currently assigned to the project.
- Eclipse. IntelliJ IDEA comes bundled with the Eclipse compiler.
- Ajc. This compiler is not included in IntelliJ IDEA distribution and should be downloaded separately.
  To use the compiler in IntelliJ IDEA, you also need to download, install, and enable the AspectJ Support plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Besides that, IntelliJ IDEA supports compilers for Flex , Groovy , Android DX . The corresponding plugins should be also downloaded, installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Build Process

Build process includes the following steps:

- Compiling source code in the source path of a module and placing results to the output path.
- Compiling source code in the test path of a module and placing results to the test output path.
- Creating copies of the resource files in the output path.
- Reporting problems in the Messages tool window.

With IntelliJ IDEA you can build your applications using Ant , Maven or Gradle .

Configuring Compiler Settings

This section describes how to access the compiler settings, modify the list of recognized resources, exclude certain paths from compilation, select the desired compiler, and configure annotation processing.

For instructions on how to configure compiler output on the project and module levels, refer to Configuring Project Compiler Output and Configuring Module Compiler Output .

To configure compiler settings

1. Open the Settings dialog box .
2. On the Compiler page , under Build, Execution, Deployment, configure common settings, for example:
   - In the Resource patterns field, update the regular expression that describes the extensions of the files to be recognized as resources. Separate individual patterns with semicolons (`;`). Refer to the list of wildcard characters , and examples.
   - If necessary, choose to perform compilation as an external process.
3. On the Excludes page, specify the files and folders that must be excluded from compilation. Click +, and choose the desired directory from the Select Path dialog box.
   Note the following:
   - If an excluded path is a dependency of the source code being compiled, this path will be included in compilation and processed by the compiler as required.
   - If a file contains errors and fails to compile, but it is not important for the current project state, or if you want to skip some files and not include them in the output directory, you can exclude such files from compilation.
4. Select a specific compiler. For example, on the Java Compiler page, select the desired compiler from the Use compiler drop-down list and specfify the compiler-specific settings.
5. In the Annotation Processors page, configure the annotation processing parameters .
6. Apply the changes and close the dialog.
7. Define the paths, providing the proper order of module roots and libraries, as described in the Configuring Module Dependencies section.

Note that the compiler in IntelliJ IDEA works sequentially, that is, in accordance with the order of module dependencies , see Order of dependencies.

Compiling Target

On this page:

- Compiling all source files in a target
- Tips and tricks

Compiling all source files in a target

To compile all source files in a specified target

1. Select a file, module, or package.
2. On the main menu, choose Build | Recompile 'target', or press `Ctrl+Shift+F9`.

Tips and tricks

When performing compilation, note that:

- When called from the editor, this method compiles the current file.
- You can compile any module separately. All modules the current one depends on will be compiled as well.

To cancel the process, do one of the following:

- Middle-click on the progress bar
- Click
- Press `Ctrl+F2`, and select the process from the pop-up menu.

Making Module

On this page:

– Making a module

– Cancelling build

Making a module

To make a module

1. Select the desired module, or any source file or directory within this module.
2. On the main menu choose Build | Make Module 'name'.

Cancelling build

To cancel the process, do one of the following:

– Middle-click on the progress bar

– Click

– Press `Ctrl+F2`, and select the process from the pop-up menu.

Making Project

On this page:

- Making a project
- Cancelling build

Making a project

To make a project, do one of the following

- On the main menu choose Build | Make Project 'name'.
- Press `Ctrl+Shift+F9`.

Cancelling build

To cancel the process, do one of the following:

- Middle-click on the progress bar
- Click
- Press `Ctrl+F2`, and select the process from the pop-up menu.

Packaging a Module into a JAR File

Before packaging a Java module into a JAR file, you need to configure a JAR artifact. See Working with Artifacts .

To build a JAR file from a module

1. On the main menu, choose Build | Build Artifact.
2. From the drop-down list, select the desired artifact of the type JAR.

   Tip The list shows all the artifacts configured for the current project. To have all the configured artifacts built, choose the Build all artifacts option.

To rebuild a project

- On the main menu choose Build | Rebuild Project.

Tip

Reviewing Compilation and Build Results

IntelliJ IDEA reports compilation and building results in the Messages tool window that displays information about the errors and warnings, providing each type of problem with its own icon and a pair of numbers that represent the row and column, where the problem occurred. In addition, the Messages tool window lets you jump from the error message to the actual location in the source code.

This section describes how to:

- Jump from an error message to the respective problem location in the source code .
- Navigate through the list of error messages .
- Save compilation report as a text file .

To jump from an error message to the problem location in the source code

Do one of the following:

- Double-click the error message.
- When Autoscroll to Source button is released, it is enough to single-click the error message.
- On the context menu of the error message choose Jump to Source.
- Press `F4`.

To navigate through the list of error messages in the Messages window

Do one of the following:

- Click or buttons in the toolbox of the Messages window.
- Press `Ctrl+Alt+Up` or `Ctrl+Alt+Down`.

To save compilation results in a text file

1. In the toolbox of the Messages window, click the Export to Text button, or press `Alt+O`.
2. In the Export Preview dialog, specify the target file, and click Save.

Tip Use the Copy button to place selected fragments of the error report to the Clipboard.

Using Multiple Build JDKs

To be able to compile an application, you need to specify a build JDK. A build JDK can be configured on a project level. If you have a complex project, you may want to configure different JDKs for different modules.

This topic explains how IntelliJ IDEA selects a build JDK if multiple JDKs are defined, and how to configure them:

- How does IntelliJ IDEA know which JDK to use?
- Configuring build JDK

How does IntelliJ IDEA know which JDK to use?

This section explains the algorithm of choosing a build JDK by IntelliJ IDEA if multiple JDKs are configured on the per-module basis. IntelliJ IDEA does the following to determine which JDK to use for compilation:

- IntelliJ IDEA checks all JDKs used in the project (i.e. the JDKs defined on both the project and module levels).
- IntelliJ IDEA calculates the latest of these JDKs. This is necessary to make sure that all modules can be compiled.
- If the version of the latest JDK configured is lower than 1.6, the JDK version used for running IntelliJ IDEA will be used. This limitation is related to the fact that the compiler API used by IntelliJ IDEA for building projects is supported starting from JDK 1.6.
- Although a specific version of the compiler will be used (in accordance with the selected JDK version), each separate module will be compiled using javac's cross-compilation feature against the libraries of the JDK defined for this particular module in the project settings.
  This protects you from a situation when a module is compiled against newer libraries than those for which dependencies are set.

Configuring build JDK

1. Open the Project Structure dialog .
2. Do one of the following:
   - To set a build JDK for an entire project, select Project in the left pane. On the right, under Project SDK, select the required SDK from the drop-down list.
   - To set a build JDK for a specific module, select Modules in the left pane, then select the required module in the central pane and specify a JDK in the Module SDK drop-down list.

     If you haven't created any projects before and specified the path to the JDK, click New, select JDK and browse for the JDK installation folder in the dialog that opens .

Note that you can also override the project- or module-level JDK settings for a particular debug or testing session by editing the appropriate Run/Debug Configuration .

Annotation Processors Support

If you use some custom annotation processors in your project to generate sources and files, validate code and produce warnings, IntelliJ IDEA enables you to easily invoke these processors as a part of the compilation process .

Annotation processing is performed on the base of the annotation profiles.

A profile is a set of configuration options for annotation processing (use of the annotation processor, location of the annotation processing output, specific annotation processors and their parameters).

The default profile always exists. All the modules comprising a project by default use this profile. When new profiles are created, modules can be associated with them. Thus, several modules can be grouped to use one type of annotation processing.

IntelliJ IDEA allows you to:

- Obtain annotation processors right from the project classpath, or from the specified location.
- Adjust the set of modules which should be covered by annotation processing of a certain profile.
- Store annotation processing results in the specified location.
- Perform annotation processing.

Configuring Annotation Processing

On this page:

- Creating annotation profile
- Associating a module with a profile
- Configuring annotation profile

To create an annotation profile

1. In the Annotation Processors page, click +.
2. In the Create new profile dialog box, specify the profile name.

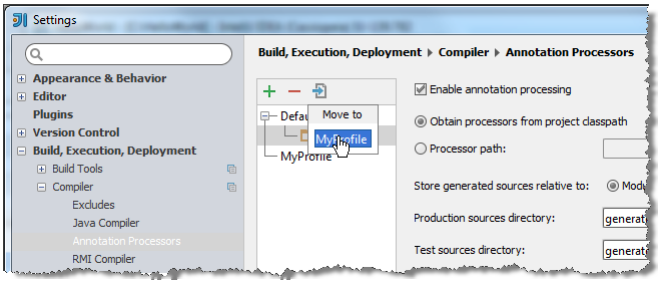
To delete a profile, select it in the list, and click —. So doing, all modules, associated with this profile, are moved to the default profile.

The default profile cannot be deleted.

To associate a module with an annotation profile

By default, all the modules of a project are associated with the default profile.

1. Select a module in the list of modules under a profile.
2. Click , or press F6.
3. From the drop-down list, select the target profile to move the selected module to.



To configure annotation processing for a profile, follow these steps

1. In the Annotation Processors page, select the desired annotation profile.
2. Enable processing annotations by selecting the corresponding check box.
3. Do the following:
   - Choose location of the annotation processors. You can opt to use annotation processor from the classpath, or specify its location by choosing it from the Select Path dialog box.
   - In the Directory name field, type the name of the directory where the annotation processor output will be stored. If the field is left blank, the files generated by the annotation processor will be stored under the project output directory. When the name is specified, the directory with this name will be created under the content root after performing annotation processing .
   - Specify the fully-qualified names of the processors to be launched. If nothing is specified, then IntelliJ IDEA will launch all the processors detected in the specified location.
   - Specify additional parameters to be passed to the annotation processors.

Working with Run/Debug Configurations

In this section:

- Run/Debug Configuration
- Creating and Editing Run/Debug Configurations
- Changing Run/Debug Configuration Defaults
- Creating and Saving Temporary Run/Debug Configurations
- Creating Folders and Grouping Run/Debug Configurations

Run/Debug Configuration

On this page:

– Basics
– Temporary configuration
– Permanent configuration
– Default run/debug configuration settings

Basics

IntelliJ IDEA enables using numerous run/debug configurations. Each run/debug configuration represents a named set of run/debug startup properties. When you perform run, debug, or test operations with IntelliJ IDEA, you always start a process based on one of the existing configurations using its parameters.

IntelliJ IDEA comes with a number of run/debug configuration types for the various running, debugging and testing issues. The user can create his/her own run/debug configurations of specific types.

Each run/debug configuration type has its own default settings. Whenever a new run/debug configuration of the respective type is created, it is based on these default settings.

Temporary configuration

A temporary run/debug configuration is automatically created every time you choose Run <item_name> or Debug <item_name> for an item without a permanent configuration. Temporary configurations can be saved as permanent.

Temporary configurations are marked with semi-transparent icons and are managed same way as the permanent configurations.

By default, 5 temporary configurations are allowed per project. You can change this limit via the Edit Configurations dialog.

Permanent configuration

A configuration of this type is explicitly created for a particular class or method. If there is no permanent configuration for an item, IntelliJ IDEA automatically creates a temporary configuration for it, when you choose Run <item_name> or Debug <item_name> on the context menu of this class or method.

Default run/debug configuration settings

The default run/debug configuration settings appear in the Run/Debug configurations dialog box under the node Defaults. They denote the settings that are used when a new run/debug configurations is created.

You can set the default settings for a specific configuration type that will become applicable to any run/debug configuration of this type created later. Changing defaults does not affect the existing run/debug configurations.

The process of editing the per-type default configuration settings is described here . The process of creating or editing the custom run-debug configurations is described on the page Creating or Editing Run/Debug configurations .

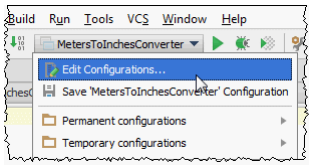
Creating and Editing Run/Debug Configurations

On this page:

– Overview
– Creating a run/debug configuration
– Editing an existing run/debug configuration

Overview

With the Navigation bar visible (View | Navigation Bar), the available run/debug configurations are displayed in the run/debug configuration selector in the Run area:



IntelliJ IDEA provides the Run/Debug Configuration dialog box as a tool for handling run/debug configurations: create configuration profiles or change the default ones.

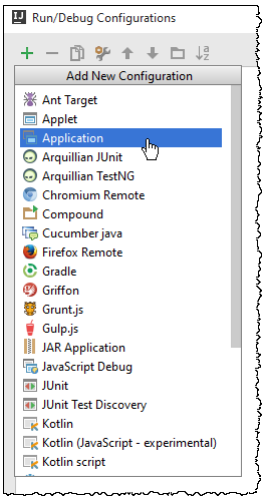
IntelliJ IDEA suggests the following ways to create a run/debug configuration:

– Create a run configuration manually on the base of the default one, using the Run/Debug Configuration dialog box.
– Save a temporary run configuration .
– Specify the target application server during module creation . IntelliJ IDEA will generate a run/debug configuration of the corresponding type.

Creating a run/debug configuration

To create a run/debug configuration, follow these steps:

1. In the Run/Debug Configuration dialog box, click + on the toolbar or press `Alt+Insert` . The drop-down list shows the default run/debug configurations. Select the desired configuration type.



   The fields that appear in the right pane, display the default settings for the selected configuration type.

   Note If you want to change the settings of the default run/debug configuration, expand the Defaults node, select the desired configuration type, and modify it as required.

2. For the new run/debug configuration:
   – Specify its name in the Name text box. This name will be shown in the list of the available run/debug configurations.
   – Specify whether you want to make IntelliJ IDEA check execution status of the instances of the same run/debug configuration. If you want to make sure that only one instance of the run/debug configuration is currently executed, select the check box Single instance only. In this case, a confirmation dialog box will show up every time you try to launch run/debug configuration, when one instance of the same type is still running.
   If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

   If this check box is not selected, you can launch as many instances of the runner as required. As the result, each runner will start in its own tab of the Run tool window.

   Tip To use an existing configuration as a pattern, create its copy by clicking the Copy button on the toolbar, then change it as required.

3. In the Before launch section, define whether you want to compile the modified sources, and run an Ant or Maven script.
4. In the Configuration tab, specify the class that contains the `main()` method, VM options, program arguments, working directory and other configuration-specific settings.
5. In the Logs tab, specify the options that control output logs produced while running or debugging application.
   In particular, specify whether IntelliJ IDEA will display the standard output and standard error output to the console.
6. For the various servers, set up the deployment and startup/connection options.
7. For the applications and tests, click the Code Coverage tab (for example, in applications run/debug configuration), and specify the options that define code coverage measurement for testing purposes.
8. Specify additional parameters depending on the configuration type. Refer to the descriptions of run/debug configuration parameters below the Run/Debug Configurations section.
9. Apply the changes and close the dialog box.

Editing an existing run/debug configuration

To change an existing run/debug configuration:

– On the main menu, choose Run | Edit Configurations.

- On the main menu, choose Run | Edit Configurations.
- With the Navigation Bar visible (View | Navigation Bar), choose Edit Configurations from the run/debug configurations selector.
- Press `Shift+Alt+F10`, then press `0` to display the Edit Configuration dialog box or select the configuration from the pop-up window and press `F4`.

In the corresponding run/debug configuration dialog box, change parameters as required.

Changing Run/Debug Configuration Defaults

On this page:

– Introduction

– Configuring defaults

Introduction

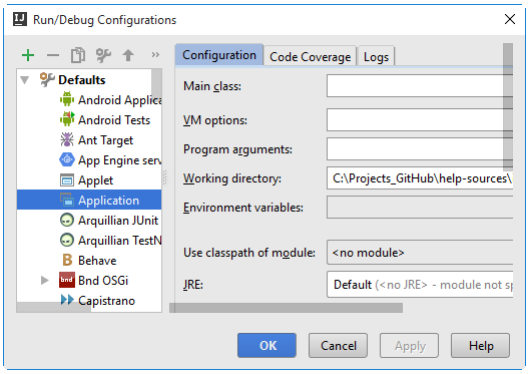
For a run/debug configuration of a particular type, you can set up the default values for one or more parameters and save them as template for further usages. In this case, next time when you create a new configuration of that type, the corresponding fields of the dialog will already contain the specified values.

Note that changing the default values does not affect the already existing run/debug configurations.

Configuring defaults

To set up the default values for a run/debug configuration, follow these steps:

1. In the left-hand pane of the run/debug configuration dialog, expand the Defaults node.
2. From the drop down list, select the desired configuration type. The corresponding configuration template appears in the right-hand pane.
3. Specify the desired parameters and click Apply to save the template.



Tip The fields Name, Share and Single instance only are not available for the default run/debug configurations.

Creating and Saving Temporary Run/Debug Configurations

On this page:

- Introduction
- Creating a temporary run/debug configuration
- Saving a temporary run/debug configuration

Introduction

Sometimes you might need to run or debug a certain class with the main() method , without creating a dedicated run configuration. In this case, IntelliJ IDEA provides a temporary run configuration .

Temporary run/debug configuration is added to the list of available configurations and works same way as the permanent run/debug configurations . You can change its settings using the Run/Debug Configuration dialog box and optionally save it as permanent.

Creating a temporary run/debug configuration

To create a temporary run/debug configuration

1. Select the desired class with the `main()` method in the Project view or open it in the editor.
2. Do one pf the following:
   - On the context menu, choose Run <name> or Debug <name>.
   - Press `Ctrl+Shift+F10`.

   IntelliJ IDEA creates a temporary configuration, which appears in the Run/Debug Configuration selector, when the run or debug session is over.

Saving a temporary run/debug configuration

To save a temporary configuration, do one of the following

- In the Run/Debug Configuration selector, choose Save <configuration name>.
- In the Run/Debug Configuration dialog box, click .
- On the context menu of the editor or Project view, choose Save <configuration name>.

Creating Folders and Grouping Run/Debug Configurations

On this page:

- Introduction
- Creating folders of run/debug configurations
- Deleting folders
- Changing order of folders

Introduction

When there are too many run/debug configurations of the same type, you can put them into directories, and thus make them visually distinguishable.

Folders area used to organize run/debug configurations. When not needed, they can be deleted, and the run/debug configurations under those folders just pass under the root of the corresponding type.

Once grouped, the run/debug configurations appear in the drop-down list under the corresponding folders:

Creating folders of run/debug configurations

To create a folder of run/debug configurations

1. Open Run/Debug Configuration dialog box.
2. In the Run/Debug Configuration dialog box, click on the toolbar. A new empty folder is created.
3. Specify the folder name in the text field to the right, or accept default.
4. Select the desired run/debug configuration of a certain type, and move under the target folder. This can be done in one of the following ways:
   - Use drag-and-drop.
   - Use the toolbar buttons .
   - Press `Alt+Up` or `Alt+Down`.
5. Apply changes. Note that the folders of run/debug configurations should not be empty. The empty folders are not saved.

Deleting folders

To delete a folder

1. In the Run/Debug Configuration dialog box, select a folder to be deleted.
2. On the toolbar, click . The selected folder is deleted silently. Any run/debug configurations grouped under this folder, are moved under the root of the corresponding type.
3. Apply changes.

Changing order of folders

To move folders up or down

1. In the Run/Debug Configuration dialog box, select one of the folders within a category of run/debug configurations.
2. Do one of the following:
   - On the toolbar, click or .
   - Press `Alt+Up` or `Alt+Down`.

   The folder in question moves one position up or down.
3. Apply changes.

After closing the dialog box, groups of run/debug configurations in the run/debug configurations selector on the main toolbar appear in the order achieved by moving folders up or down within a category.

Running

This section describes the procedures that are common for the various types of applications:

- Running Applications
- Rerunning Applications
- Reviewing Results
- Stopping and Pausing Applications
- Setting Configuration Options
- Setting Log Options

For the details related to running applications in the supported frameworks, refer to Language and Framework - Specific Guidelines

Running Applications

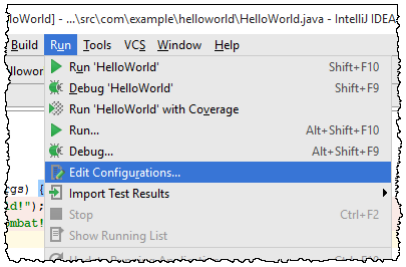
On this page:

- Introduction
- Running an application
- Running a class with a main() method

Introduction

IntelliJ IDEA enables running entire applications as well as classes with a main() method .

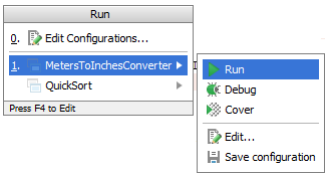
IntelliJ IDEA makes use of the settings defined in a Run/Debug Configuration . All the run configurations that exist in a project, are available in the Select Run/Debug Configuration drop-down list.



Running an application

To run an application

- Do one of the following:
  - In the left gutter, click the icon ▶, and choose the desired command.
  - On the main toolbar, select the desired run configuration, and:
    - Choose Run | Run from the main menu.
    - Click ▶.
    - Press `Shift+F10`.
  - Press `Shift+Alt+F10`, select the desired run configuration from the pop-up menu, and press `Enter`.



From this pop-up menu you can:

- Invoke the Edit Configuration dialog.
- Edit the selected configuration before launch (`F4`).
- Instantly delete a temporary configuration (`Delete`).
- Switch from run to debug and vice versa (hold `Shift`).
- Access a previously selected configuration (`1`).
- Access context-dependent configuration (`2` or `3`).

This pop-up menu can also be quickly accessed by pressing `F9`, when you're not running any debug session.

Note If the options that launch build or before running were enabled, IntelliJ IDEA runs the build , and after success will run the application. Otherwise, the program will start immediately.

Note that after you've started a run session, the ▶ icon that marks the Run tool window and in the Run/Debug Configuration Selector toggles to ▶ to indicate that the run process is active.

Running a class with a main() method

To run a class with a main() method

- Open the class in the editor and do one of the following:
  - In the left gutter, click the icon ▶, and choose the desired command.
  - Choose Run <method name > on the context menu
  - Press `Ctrl+Shift+F10`
- Select the class in the Project tool window and choose Run <method name > on the context menu of the selection.

Rerunning Applications

You can re-run an application if its tab is still opened in the Run window. The program re-runs with the initial settings.

To re-run an application

1. In the Run window, select the tab where the desired application is opened.
2. In the toolbar of the Run window, click the Rerun button , or press `Ctrl+F5`.

   Tip If you want to re-run without loosing focus in the editor tab you are working in, press `Shift+F10`.

Reviewing Results

You can review any output from your running applications in the Run window console. The output from each application is displayed in its own tab of the Run tool window, named after the corresponding run/debug configuration .

If you re-run an application, the new output overwrites the contents of the tab. To preserve the output of an application, even if you re-run it, pin the output tab.

Stopping and Pausing Applications

In the Run tool window, you can stop a program, or pause its output. If a program is stopped, its process is interrupted and exits immediately. When program output is paused, the program continues running in the background, but its output is suspended.

To stop a program, do one of the following

- In the Run tool window, click the Stop button on the toolbar, or press `Ctrl+F2`.
- To close the active tab, click the Close button, or press `Ctrl+Shift+F4`.

To suspend program output

- In the Run tool window, click the Pause button on the toolbar.

Setting Configuration Options

Configuration options include VM settings, arguments that should be passed to the program, working directory, classpath and SDK. Refer to the Run/Debug Configuration dialog for detailed description of the fields.

To define Configuration options of a run/debug configuration

1. Click Configuration tab of the Edit Run/Debug Configurationdialog.
2. In the Main class field, specify the class that contains the `main()` method. To do that, type the fully qualified name manually, or click the ellipsis button and select the desired class from the Choose Main Class dialog.

   In the Choose Main Class dialog, you can locate the desired class using one of the following ways:
   - Click the Project tab, and select class with the `main()` method from the project tree view.
   - Click the Search by Name tab and start typing the class name. As you type, the list of available classes narrows down to match your entry.

   Click OK, or press `Enter` when ready.
3. In the VM options field, type optional VM arguments, for example the heap size, garbage collection options, file encoding, etc. If the line of VM arguments is too long, click and type the text in the editor dialog.
4. In the Program parameters field, type optional list of parameters that should be passed to the `main()` method through the array of its arguments.
5. In the Working directory field, specify the current directory that your application will use while running.
6. In the Use classpath and SDK of module field, select the desired module from the list of modules existing in the project.

Setting Log Options

Use Logs tab in the Run/Debug Configuration dialogs to configure the way log files, generated by an application or server, are displayed in the console.

If your application or server generates log files, the default entries will be automatically added to the log file list in the Run/Debug Configuration dialog.

To configure Logs options

1. In the Run/Debug Configuration dialog box, click Logs tab. The table Log files to be shown in console displays the list of log files (if any).
2. Click +. Edit Log Files Aliases Dialog dialog is displayed.
3. In the Alias field, type the alias name to show in the list of log entries. In the Log File Location field, type the fully qualified name of the log file, or specify its location by pressing the ellipsis button. Select whether you want to show all or last file coverable by pattern. Click OK to close the dialog.
4. Activate the log entry. To do that, select the check box in the Is Active column.
5. To skip the previous content, select the check box in the Skip Content column.

Note If you are using third-party logging tools, you might want to make the message's output that mimics a standard linkage to the source code as for stacktrace line (at `<fully-qualified-class-name>.<method-name>(<file-name>:<line-number>)`). For that, add specific Conversion Pattern to your log.xml configuration file.

For example, in a log4j Conversion Pattern this would be `<param name="ConversionPattern" value="%-5p - [%-80m] - at %c.F:n"/>`.

Debugging

- Debugging
  - Overview
  - General debugging steps
- Breakpoints
- Using Breakpoints
- Configuring Debugger Options
- Starting the Debugger Session
- Pausing and Resuming the Debugger Session
- Reloading Classes
- Examining Suspended Program
- Exploring Frames
- Customizing Views
- Finding the Current Execution Point
- Stepping Through the Program
- Monitoring the Debug Information
- Inline Debugging
- Attaching to Local Process
- Tutorial. Debugging with Chronon

Overview

This section describes the procedures that are common for various types of applications.

For details on debugging applications in the supported frameworks, refer to Language and Framework - Specific Guidelines .

IntelliJ IDEA provides a full range of facilities for debugging your source code:

- Breakpoints in Java.
- Breakpoints in JavaScript.
- Customizable breakpoint properties: conditions, pass count, etc.
- Frames , variables , and watches views in the debugger UI.
- Runtime evaluation of expressions .

General debugging steps

1. Configure the dependencies and libraries to be passed to the compiler and generate the debugging information.
2. Configure common debugger behavior, including stepping speed , class reloading policy , or scrolling of the editor canvas .
3. Configure the debugger options .
4. To debug CoffeeScript, TypeScript, and Dart code, you need source maps generated in addition to the JavaScript code.
   Source maps set the correspondence between lines in your original code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.

   JavaScript and source maps are generated by transpiling the original code manually using the File Watcher of the corresponding type (CoffeeScript, TypeScript, or Dart). After that, you can debug the output JavaScript code.

   For details, see Using File Watchers , Transpiling CoffeeScript to JavaScript , and Transpiling TypeScript to JavaScript .
5. Define a run/debug configuration for the application to be debugged.
6. Create breakpoints in the source code.
7. Launch a debugging session.
8. Pause or resume the debugging session as required.
9. During the debugger session, step through the breakpoints , evaluate expressions , change values on-the-fly , examine suspended program , explore frames , set watches , reload classes , and customize views .

After you've started a debug session, the icon that marks the Debug tool window toggles to to indicate that the debug process is active.

If you want to see a list of all currently running applications, select Run | Show Running List from the main menu. A popup listing all active applications will be displayed in the top-right corner of the editor.

Note that IntelliJ IDEA lets you debug decompiled code in the same way as your normal source files, provided that it contains line number attributes.

Breakpoints

Breakpoints are source code markers used to trigger actions during a debugging session.

In this part:

– Types of Breakpoints
– Breakpoints Icons and Statuses

Types of Breakpoints

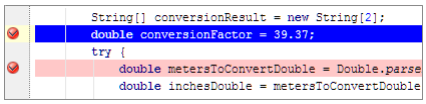
On this page:

- Introduction
- Line Breakpoint
- Temporary Line Breakpoint
- Method Breakpoint
- Exception Breakpoint
- Field Watchpoint
- JavaScript / Flex /PHP Breakpoints

Introduction

IntelliJ IDEA lets you create breakpoints of several types. Each breakpoint type supported by IntelliJ IDEA addresses different debugging needs and has its own individual settings.

Breakpoints are triggered when the program reaches the specified line of source code, before it is executed. The line of code that contains a set breakpoint, is marked with a red stripe; once such line of code is reached, the marking stripe changes to blue.



Once set, a breakpoint remains in project until removed. Breakpoints can only be set on executable lines of code. Comments, declarations of fields or methods, and empty lines are not valid locations for breakpoints.

Tip If a file with breakpoints has been modified externally, for example, updated from a version control repository, or changed in an external editor, so that line numbers are changed, then the breakpoints will be moved accordingly. Note that IntelliJ IDEA should be running at the moment of such modification; otherwise, such changes will pass unnoticed.

Line Breakpoint

These breakpoints are assigned to lines of source code and are used to target a particular section for debugging.

Temporary Line Breakpoint

These breakpoints are assigned to lines of source code and are used to target a particular section for debugging. When hit, such breakpoints are immediately removed.

Method Breakpoint

Method breakpoints act in response to the program entering or exiting a particular method. They let you target your debugging sessions by method you wish to investigate, rather than by line number. Method breakpoints let you follow the program flow at the method level as well as check entry and exit conditions. Note that using method breakpoints can slow down the application you are debugging.

Exception Breakpoint

IntelliJ IDEA provides exception breakpoints for Java and JavaScript.

Exception breakpoints are triggered when the specified exception is thrown. Unlike the line breakpoints, which require specific source references, exception breakpoints apply globally to the exception condition, rather than to a particular code reference.

Field Watchpoint

Field watchpoints allow you to react to any access or modification of specific instance variables. For example, if at the end of a complicated process you are ending up with an obviously wrong value on one of your fields, then setting up a field watchpoint may be the quickest way to determine the origin of the fault.

JavaScript / Flex /PHP Breakpoints

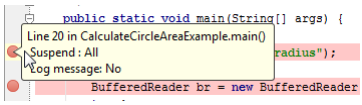
JavaScript, Flex, and PHP breakpoints are identical to line breakpoints in Java.

These breakpoints are assigned to particular lines of JavaScript or PHP source code. They can be set in `*.html` files as well as in `*.js` or `*.php` files and are used to target a particular section of code for debugging.

Breakpoints Icons and Statuses

When a breakpoint is set, the editor displays a breakpoint icon in the gutter area to the left of the affected source code. A breakpoint icon denotes status of a breakpoint, and provides useful information about its type, location, and action.

The icons serve as convenient shortcuts for managing breakpoints. Clicking an icon removes the breakpoint. Successive use of `Alt` - click on an icon toggles its state between enabled and disabled. The settings of a breakpoint are shown in a tooltip when a mouse pointer hovers over a breakpoint icon in the gutter area of the editor, or by right-clicking an icon.



The table below summarizes the possible breakpoint states:

Status	Line	Temporary Line	Exception	Method	Field	Description
	JavaScript					
	Flex					
Enabled						Shown at design-time or during the debugging session when the class with such breakpoint is not yet loaded.
Valid			N/A	N/A		Shown at run-time when the breakpoint is recognized by the debugger as set on an executable code line.
Invalid				N/A		Shown when the breakpoint is set on a commented or non-executable line indicating that such breakpoint would not be hit.
Disabled						Indicates that nothing happens when the breakpoint is hit.
Conditionally disabled						This state is assigned to breakpoints when they depend on another breakpoint to be activated.

Using Breakpoints

In this section:

- Accessing Breakpoint Properties
- Configuring Breakpoints
- Creating Line Breakpoints
- Creating Exception Breakpoints
- Creating Field Watchpoints
- Creating Method Breakpoints
- Enabling, Disabling and Removing Breakpoints
- Moving Breakpoints
- Named Breakpoints
- Navigating Back to Source
- Working with Groups of Breakpoints

Accessing Breakpoint Properties

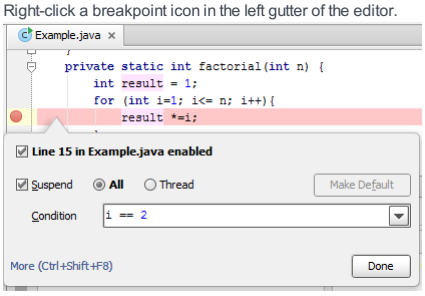
To view the whole list of the breakpoints contained in an application , use the Breakpoints dialog box. For each individual breakpoint in the list, you can view and change its properties as required.

To view the list of all breakpoints and their properties, do one of the following:

- On the main menu, choose Run | View Breakpoints.
- Press `Ctrl+Shift+F8`.
- In the toolbar of the Debug tool window , click .
- Breakpoints are visible in the Favorites tool window.

To view properties of a single breakpoint

- Right-click a breakpoint icon in the left gutter of the editor.



Configuring Breakpoints

On this page:

- Basics
- Configuring breakpoints

Basics

For a breakpoint, you can configure the following properties:

- Actions to be performed upon hitting a certain breakpoint.
- Suspend policy, which defines whether the application should be suspended upon hitting the breakpoint.
- Dependencies on other breakpoints.
- Conditions defining when a breakpoint is hit.

IntelliJ IDEA suggests the following way to change the breakpoints properties:

- Using the Breakpoints dialog box, for a breakpoint selected in the list.
- Using breakpoint icon in the left gutter

Configuring breakpoints

To configure actions, suspend policy and dependencies of a breakpoint

1. Do one of the following:
   - Right-click a breakpoint in the left gutter, and then click the link More or press `Ctrl+Shift+F8`.
   - Open the Breakpoints dialog box as described on page Accessing Breakpoint Properties and select the desired breakpoint in the list.
   - In the Favorites tool window, select the desired breakpoint, and click .

   Note that the pop-up window shows less options than the Breakpoints dialog box. To show hidden options, click More.

2. Define the actions to be performed by IntelliJ IDEA on hitting breakpoint:
   - To notify about the reaching of a breakpoint with a text message in the debugging console, select the Log message to console check check box.
     To evaluate an expression in the context of a breakpoint and display its value in the debugging console, check the option Evaluate and log, and enter a valid expression in the option field.

     This feature lets you obtain information about your running application without having to suspend its execution.
   - To set a breakpoint the current one depends on, select it from the Disabled until selected breakpoint hit drop-down list. Once dependency has been set, the current breakpoint is disabled until selected one is hit.
     - Choose Disable again radio button to disable the current breakpoint after selected breakpoint was hit.
     - Choose Leave enable radio button to keep the current breakpoint enabled after selected breakpoint was hit.
   - Enable suspending an application upon reaching a breakpoint by selecting the Suspend check box, and then select one of the option buttons to specify the way a running program will be paused. For more information on the Suspend options, refer to Breakpoints dialog reference.
   - To set the break condition, enable condition by selecting the appropriate check box, and enter the desired expression in the Condition field.
     If the expression evaluates to true, the user-selected actions are performed. If the evaluation result is false, the breakpoint does not produce any effect.

3. The following options are defined in the Breakpoints dialog box (if you edit properties of a particular breakpoint, click More):
   - To limit breakpoint hits only with particular object instances using instance IDs, check the Instance filters option and type the instance ID value, or click the ellipsis button and specify instance ID in the Instance Filters dialog.
   - To define breakpoint behavior with regards to particular classes, select the Class Filter check box and specify the class filter. Type the class filter manually or click the Browse button and create the class filter definition in the Class Filters dialog box that opens.
   - To define the number of times a breakpoint is reached but ignored, select the Pass count check box and specify the number of passes the breakpoint should be skipped before it is hit.
     For more information, refer to the Breakpoints dialog reference.

Creating Line Breakpoints

On this page:

- Basics
- Creating line breakpoints in the editor
  - Important notes
- Creating temporary line breakpoints
- Deleting line breakpoints

Basics

A line breakpoint is a breakpoint assigned to a specific line in the source code.

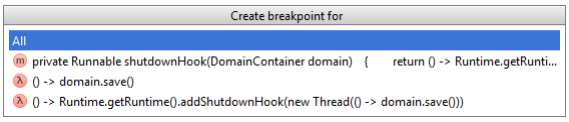
Line breakpoints can be set on executable lines. Comments, declarations and empty lines are not valid locations for the line breakpoints.

Creating line breakpoints in the editor

1. Place the caret on the desired line of the source code.
2. Do one of the following:
   - Click the left gutter area at a line where you want to toggle a breakpoint.
   - On the main menu, choose Run | Toggle Line Breakpoint.
   - Press `Ctrl+F8`.

Important notes

- For the lambda expressions, you can set multiple breakpoints within a single line. To do this, click the left gutter area at a line where you want to set a breakpoint. A drop-down list will appear where you can select which method(s) you want to set a breakpoint at:



  IntelliJ IDEA highlights each of the lambda expressions as you move the mouse over the options.
- If you want to set a line breakpoint in the default class constructor, it is enough to set a line breakpoint on the first line of this class, since the default constructor is mapped to it:

```
class A { //
                    set a breakpoint on this
                    line
                    }
```

- When one sets a breakpoint on a folded method , a line breakpoint is set on the first executable line after method declaration.

Creating temporary line breakpoints

1. Place the caret on the desired line of the source code.
2. Do one of the following:
   - On the main menu, choose Run | Toggle Temporary Line Breakpoint.
   - Press `Ctrl+Shift+Alt+F8`.

Deleting line breakpoints

Do one of he following:

- In the Breakpoints dialog box, select the desired line breakpoint, and click .
- In the editor, locate the line with the line breakpoint to be deleted, and click its icon in the left gutter.
- Place caret on the desired line and press `Ctrl+F8`.

Creating Exception Breakpoints

1. On the main menu, choose Run | View Breakpoints, or press `Ctrl+Shift+F8`.
2. In the Breakpoints dialog box that opens, click +.
3. Select Java Exception Breakpoints or JavaScript Exception Breakpoint from the drop-down list.
4. In the Choose Exception Class dialog box, specify the desired exception class from the library, or from the project, and click OK.
   IntelliJ IDEA returns you to the Breakpoints dialog box.
5. Configure the new exception breakpoint as described in Configuring Breakpoints .

Creating Field Watchpoints

Field watchpoints help you target your debugging search to specific instance variables.

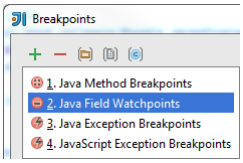
On this page:

– Creating field watchpoints using the Breakpoints dialog
– Creating field watchpoints using the editor
– Creating field watchpoints from the Debug tool window
– Deleting field watchpoints

Creating field watchpoints using the Breakpoints dialog

To create a field watchpoint using the Breakpoint dialog

1. On the main menu, choose Run | View Breakpoints, or press `Ctrl+Shift+F8`.
2. In the Breakpoints dialog box that opens, click +.
3. Select Field Watchpoint from the drop-down list:



4. In the Add Field Watchpoint dialog box that opens, specify the following:
   – Fully qualified name of a class that contains the desired field. You can type it manually, or click ..., and find the desired class by name, or from the project.
   – Field name. You can type it manually, or click ... and select the desired field from the list of fields in the selected class.

Creating field watchpoints using the editor

To create a field watchpoint from the editor

1. Open the desired class in the editor, and locate the field you want to create a watchpoint for.
2. `Alt+click` on the left gutter at the field declaration line.

Creating field watchpoints from the Debug tool window

To create a field watchpoint from the Debug tool window

1. During the debugging session, open the Variables tab.
2. Select the desired field and choose Add Field Watchpoint on the context menu.

Deleting field watchpoints

To delete a field watchpoint

1. In the Breakpoints dialog box, select the desired field watchpoint, and click –.
2. In the editor, locate the line with the watchpoint to be deleted, and click its icon in the left gutter.

Creating Method Breakpoints

On this page:

– Introduction

– Creating method breakpoints

– Deleting method breakpoints

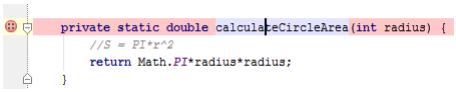
Introduction

Method breakpoints let you follow the program flow at the method level.

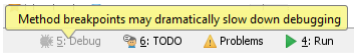
Creating method breakpoints

To create a breakpoint using the editor

1. Place the caret inside the method where you want to toggle a method breakpoint.
2. On the main menu, choose Run | Toggle Method Breakpoint. Method breakpoint appears at the method declaration.



A balloon appears, informing you about the possible slow-down of the debugging process:



Alternatively, just click the left gutter at the method declaration.

To create a method breakpoint using the Breakpoints dialog

1. On the main menu, choose Run | View Breakpoints, or press `Ctrl+Shift+F8`.
2. In the Breakpoints dialog box that opens, click +.
3. Select Method Breakpoint from the drop-down list.
4. In the Add Method Breakpoint dialog box, specify the class name pattern, including the package name, and the name of the desired method.

When a debugging session starts, the application will pause in all classes with the names matching the specified pattern, at the specified method.

Deleting method breakpoints

To delete a method breakpoint

1. Click the method breakpoint icon in the left gutter.
2. On the main menu, choose Run | Toggle Method Breakpoint.

Enabling, Disabling and Removing Breakpoints

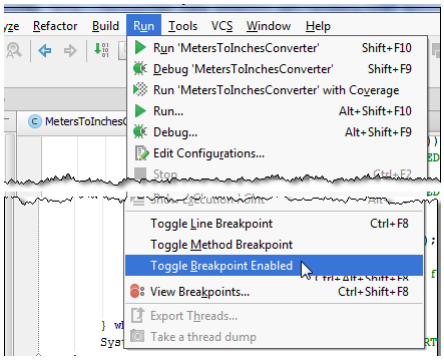
On this page:

- Toggling between the enabled and disabled state of a breakpoint
- Disabling a breakpoint temporarily in the editor

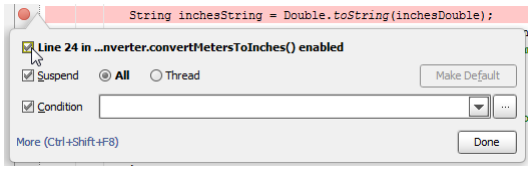
Toggling between the enabled and disabled state of a breakpoint

When you temporarily disable or enable a breakpoint, its icon changes from to and vice versa.

1. Place the caret at the desired line with a breakpoint.
2. Do one of the following:
   - On the main menu, choose Run | Toggle Breakpoint Enabled:



   - Right-click the desired breakpoint icon, select or deselect the <breakpoint name> enabled check box, and then click Done.



   - `Alt` -click on the breakpoint icon.

Tip Alternatively, open the Breakpoints dialog box, select the desired breakpoint, and select or clear the check box to its left.

Disabling a breakpoint temporarily in the editor

When you temporarily disable a breakpoint, its icon changes from to .

Alternatively, open the Breakpoints dialog box, as described on page Accessing Breakpoint Properties , select the desired breakpoint, and clear the check box next to it or the Line <line number> in <file name> check box in the right-hand pane.

Enabling a temporarily disabled breakpoint in the editor

When you enable a temporarily disabled breakpoint, its icon changes from to .

1. Place the caret at the desired line with a breakpoint.
2. Do one of the following:
   - Right-click the desired breakpoint icon, select the Line <line number> in <file name> check box in the pop-up dialog box that opens, and then click Done.
   - With the `Alt` key pressed, click the breakpoint icon.

Alternatively, open the Breakpoints dialog box, as described on page Accessing Breakpoint Properties , select the desired breakpoint, and select the check box next to it or the Line <line number> in <file name> check box in the right-hand pane.

Removing a breakpoint

To remove a breakpoint permanently, do one of the following:

- Open the Breakpoints dialog box , select the desired breakpoint, and click .
- Click the breakpoint icon in the left gutter of the editor.

Removing all breakpoints of a certain type

1. On the main menu, choose Run | View Breakpoints, or press `Ctrl+Shift+F8`.
2. In the Breakpoints dialog, press the left arrow key to select the desired category.
3. Press `Delete`.

All breakpoints of a certain type will be deleted.

Moving Breakpoints

To move a breakpoint, drag-and-drop it to the target line.

Note Field/Method breakpoint can be dragged to another field/method declaration only.

Named Breakpoints

On this page:

– Introduction

– Editing breakpoint description

– Searching for a breakpoint using its name

Introduction

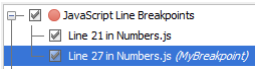
IntelliJ IDEA makes it possible to add a name or a short description to a breakpoint to facilitate search.

Editing breakpoint description

To edit a breakpoint description

1. Open the Breakpoints dialog .
2. Right-click a breakpoint you are interested in.
3. On the context menu, choose Edit description.
4. In the Edit Description dialog box, type the desired description.

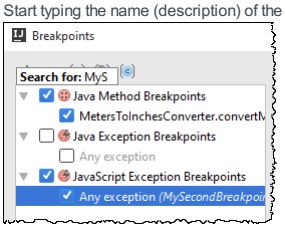
The specified description shows in italic next to the address of a breakpoint in the Breakpoints dialog:



Searching for a breakpoint using its name

To search a brekpoint by name

1. Open the Breakpoints dialog .
2. Start typing the name (description) of the desired breakpoint.



The breakpoint with the matching description gets the focus.

To navigate to the breakpoints source from the Breakpoints dialog box

- To view the selected breakpoint without closing the dialog box, use the preview pane.
- To open the file with the selected breakpoint for editing, double-click the desired breakpoint. To close Breakpoints dialog, press `F4`. The caret will be placed at the line marked with the breakpoint in question.

Working with Groups of Breakpoints

On this page:

- Introduction
- Creating groups of breakpoints
- Moving breakpoints to another group, or out of a group
- Toggling a group of breakpoints

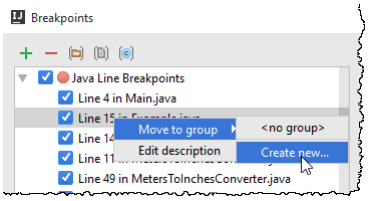
Introduction

IntelliJ IDEA makes it possible to organize breakpoints in groups, for example, to mark out breakpoints for a specific problem. This is done in the Breakpoints dialog .

Creating groups of breakpoints

To create a group of breakpoints

1. In the Breakpoints dialog , right-click one or more breakpoints you are interested in.
2. On the context menu, point to the command Move to group, and then on the submenu, choose Create new...:

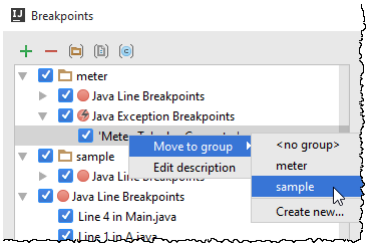


3. In the New Group dialog box, type the name of the new group. The selected breakpoint moves to the newly created group.
4. Optionally, you can right-click a group of breakpoints and select Set as default from the popup menu. All newly created breakpoints will be automatically added to this group.

Moving breakpoints to another group, or out of a group

To move breakpoints to another group

1. In the Breakpoints dialog , right-click one or more breakpoints you are interested in.
2. On the context menu, point to the command Move to group, and then on the submenu, choose the desired group name:



The breakpoints in question move to the selected group.

To move breakpoints out of a group

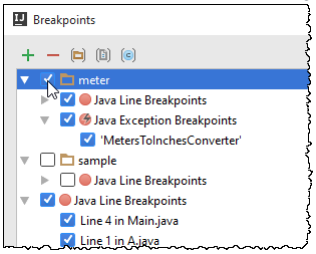
1. In the Breakpoints dialog , right-click one or more breakpoints within a group.
2. On the context menu, point to the command Move to group, and then on the submenu, choose <no group>.
3. The selected breakpoints move to a node according to their type .

Toggling a group of breakpoints

Using groups of breakpoints, it is possible to toggle all breakpoints within a group in a single click.

To toggle all breakpoints belonging to a certain group

- Select or clear the check box to the left of a group name:



Configuring Debugger Options

On this page:

- Introduction
- Configuring debugger settings

Introduction

IntelliJ IDEA supports debugging for Java and Groovy applications, classes, and files. The debugging functionality is incorporated in IntelliJ IDEA, you only need to configure its settings.

Depending on the plugins enabled, IntelliJ IDEA can also support debugging for other languages, for example, JavaScript , Flex or PHP .

IntelliJ IDEA supports local and remote JavaScript debugging only in Google Chrome and other browsers of the Chrome family.

Configuring debugger settings

To configure settings required for debugging, perform the following general steps

1. In the Project Structure , configure the roots, dependencies and libraries to be passed to the compiler .
2. In the Settings/Preferences dialog box, configure the debugger options:
   - Under the Build, Execution and Deployment section, point to Compiler node, click Java Compiler , or RMI Compiler , and select the check box Generate debugging info.
   - Under the Build, Execution and Deployment, click Debugger , and define the debugger options as required.

Starting the Debugger Session

On this page:

- Before debugging
- Debugging an application

Before debugging

- Configure debugger options .
- Specify the roots, dependencies and libraries to be passed to the Debugger .
- Set breakpoints in the source code.
- If necessary, create or modify the corresponding Run/Debug configuration .

The debug session starts with the selected run/debug configuration. Note that several debug processes can be launched simultaneously.

When debugging an application in IntelliJ IDEA, keep in mind that

- If the Make module before running/debugging/reloading option has been selected, IntelliJ IDEA first compiles all modified sources in your project.
- IntelliJ IDEA proceeds with debugging, if compilation reports no errors.
- If the code has not been compiled before debugging, the source and class files might be out of sync.
- If you specify the `-classpath` option in the VM Options field, the selected module classpath will be overridden.
- If you debug a JavaScript source, IntelliJ IDEA opens a browser for the HTML file with your script in a separate frame.

Debugging an application

To start debugging an application, do one of the following

- Select the run/debug configuration to execute, and then do one of the following:
  - Click icon in the left gutter, and then choose .
  - Click on the toolbar.
  - Choose Run | Debug on the main menu.
  - Press `Shift+F9`.
- Press `Shift+Alt+F9`, select the configuration from the pop-up menu, and press `Enter`.

Note that after you've launched a debug session, the icon that marks the Debug Tool Window toggles to to indicate that the debugging process is active.

Pausing and Resuming the Debugger Session

On this page:

- Introduction
- Pausing the debugger session
- Resuming the debugger session

Introduction

When a breakpoint is hit, or when a running thread or an application is paused manually, the debugging session is suspended.

Pausing the debugger session

To pause the debugger session, do one of the following

- On the main menu, choose Run | Pause Program.
- Click on the Debug toolbar.

Resuming the debugger session

To resume the debugger session, do one of the following

- On the main menu, choose Run | Resume Program.
- Click on the Debug toolbar.
- Press F9.

Reloading Classes

On this page:

- Introduction
- Reloading changed classes
- Configuring reloading behavior

Introduction

You can reload classes changed during debugging without need to restart the entire application using the HotSwap mechanism.

At the moment due to original limitations of Java SDK the HotSwapping is possible ONLY if a method body is altered. In all other cases (like changing method or class signature), the class reload is impossible and the corresponding error message appears.

Reloading changed classes

To reload changed classes

1. Do one of the following:
   - On the main menu, choose Run | Reload Changed Classes.
   - On the main menu, choose Build | Compile "class_name" to recompile an altered class during debug.
2. In the Reload Changed Classes dialog box, confirm reloading. Results are displayed in the Messages tool window .

Configuring reloading behavior

To configure reloading behavior

1. On the main menu, choose File | Settings , and under Build, Execution, Deployment expand the Debugger node.
2. Open HotSwap page.
3. Click one of the radio buttons in the group Reload classes after compilation. You can opt to always reload classes, reload after confirmation, or never do it.

Examining Suspended Program

On this page:

– Basics
– Examining a suspended thread
– Navigating between frames
– Exporting threads

Basics

When a breakpoint is hit, or a program execution is manually suspended , you can examine your application by analyzing frames.

A frame corresponds to an active method or function call. A frame stores the local variables of the called method or function, the arguments to it, and the code context that enables expression evaluation.

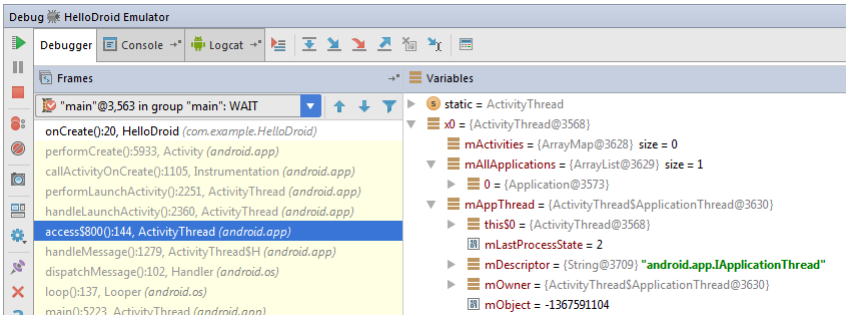
All currently active frames are displayed on the Frames pane of the Debug tool window, where you can switch between them and analyze the information stored therein.

Examining a suspended thread

To examine frames of a suspended thread

1. Select a thread from the thread selector drop-down list on top of the Frames pane. The list of frames is displayed:

2. Select a frame from the Frames list. The Variables pane shows all the variables available to the method call in this frame, so you can further explore them.



Navigating between frames

Do one of the following:

– Use up and down arrow buttons on the toolbar.
– Use `Up` and `Down` shortcuts.

You do not need to perform any actions to navigate to the frame's source code. IntelliJ IDEA automatically jumps to the source code of the selected frame in the editor.

Exporting threads

If you need to get a report on the status of all the threads, you can export threads information.

To export threads

1. Right-click anywhere in the Frames tab and select Export Threads from the context menu, or select Run | Export Threads from the main menu.
2. To save a report as a text file, specify the path to the file in the Export Threads dialog and click Save.
3. To copy it to the Clipboard, click Copy.

Exploring Frames

When a frame is selected in the list, all the values available to this frame's method call are displayed in the Variables pane of the Debug tool window, so you can further explore them. This section describes the ways to simplify examining these values:

- Evaluating Expressions
- Adding, Editing and Removing Watches
- Inspecting Watched Items
- Setting Labels to Variables, Objects and Watches
- Navigating to Source Code from the Debug Tool Window

Evaluating Expressions

On this page:

- Basics
- Limitations
- Evaluating Expressions or Code Fragments in a Stack Frame
- Evaluating Arbitrary Expressions
- Evaluating Expressions in the Editor

Basics

IntelliJ IDEA enables you to evaluate expressions and code fragments in the context of a stack frame currently selected in the Frames pane of the Debug tool window .

In addition to regular expressions, you can also evaluate operator expressions, lambda expressions, and anonymous classes.

The following evaluation modes are available:

- Expression Mode for evaluating single-line expressions.
- Code Fragment Mode for evaluating short code portions. You can evaluate declarations, assignments, loops and `if/else` .

Besides, IntelliJ IDEA provides a way to quickly evaluate an expression in the editor at caret or a selection.

Limitations

While using the Expression Evaluation feature, be aware of the following:

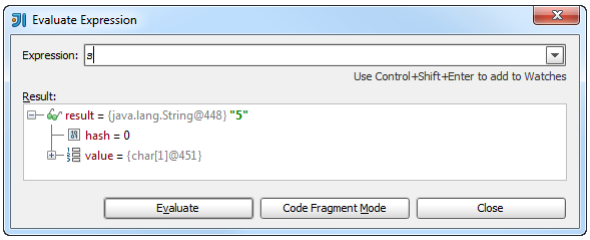
- A method can be invoked within the Expression Evaluation dialog only if the debugger has stopped at a breakpoint, but has not been paused.
- Expression Evaluation can only be "single-level". In other words, if IntelliJ IDEA stops at a breakpoint within a method called from the Expression Evaluation, you cannot use the Expression Evaluation feature again.
- If a method invoked within Expression Evaluation has a breakpoint inside its body, this breakpoint will be ignored.

Tip Note that in certain operating systems the key and mouse combinations may not work as described here. In this case, it's necessary to tweak the operating system's keymap. For example, if you are using Ubuntu, mind the windows manager, whose shortcuts conflict with that of IntelliJ IDEA.

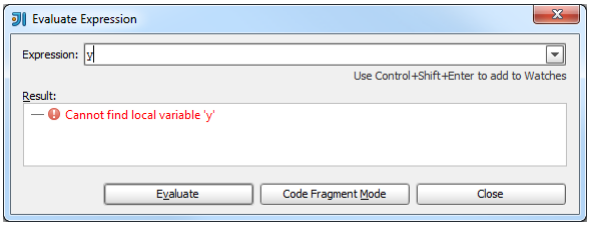
Evaluating Expressions or Code Fragments in a Stack Frame

To evaluate an expression or a code fragment in a stack frame, do the following:

1. In the Frames pane, select the thread where you want an expression to be evaluated.
2. Invoke the Evaluate Expression command in one of the following ways:
   - On the main menu, choose Run | Evaluate Expression
   - On the context menu of the editor, choose Evaluate Expression
   - Press `Alt+F8`
   - Click on the Stepping toolbar in the Debug tool window .
3. Select an evaluation mode. If you want to evaluate a code fragment, click the Code Fragment Mode button.
4. Depending on the selected mode, type the expression or statements to evaluate in the text field and click Evaluate.



If the specified expression cannot be evaluated, the possible reason will be briefly described in the Result pane of the dialog box.



Tip If you have assigned a label to a variable, object, or watch, you can reference it by this label as if it were a local variable `<label-name>_DebugLabel` defined in the same context where the expression is evaluated. IntelliJ IDEA also displays this label in the completion suggestion list.

Evaluating Arbitrary Expressions

1. Open the Evaluate Expression dialog box in one of the following ways:
   - Choose Run | Evaluate Expression on the main menu.
   - Press `Alt+F8` .
   - To evaluate a specific variable, select it on the Variables pane of the Debug tool window , then choose Run | Evaluate Expression or press `Alt+F8` .
2. In the Evaluate Expression dialog box, specify the expression you want to evaluate. Do one of the following:
   - In the Expression field, type the expression in question or choose one of the previously evaluated expressions from the drop-down list.

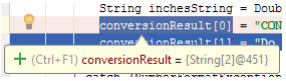
     If you have selected a specific variable on the Variables pane, this variable will be displayed in the Expression text box.
   - To evaluate a code fragment, click the Code Fragment Mode button and fill in the Code Fragment text box.

To return to the original mode, click the Expression mode button.

3. Click the Evaluate button. The Result read-only field shows the evaluation output. If the specified expression cannot be evaluated, the Result field explains the reason.

Evaluating Expressions in the Editor

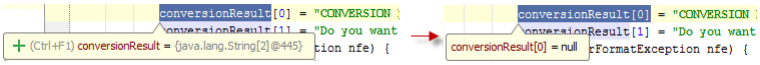
During a debugger session, the value of any expression is shown in the tooltip every time you hover your mouse pointer over it. If an expression contains children, clicking + expands the node and displays all children.



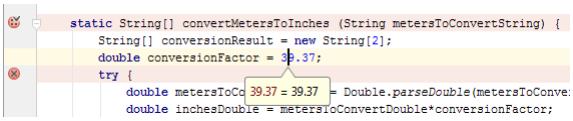
You can also use the Quick Evaluate expression functionality that lets you view the value of an expression using the keyboard only.

There are two ways to evaluate an expression quickly:

1. By using the Show value tooltip on code selection functionality:
   a. In the Debugger | Data Views settings page, enable the Show value tooltip on code selection option.
   b. Select a code fragment with the mouse, or by clicking `Ctrl+W`. A tooltip with the expression value automatically appears under the selection and changes each time you change the selection:



2. By manually invoking the tooltip with the expression value:
   a. Place the caret at the desired location, or select an expression to be evaluated.
   b. Choose Run | Quick Evaluate Expression on the main menu, or press `Ctrl+Alt+F8`. The tooltip with the expression value appears under the selected expression.



Adding, Editing and Removing Watches

On this page:

- Introduction
- Creating watches
- Editing watches
- Deleting watches

Introduction

If you want to evaluate a number of variables or expressions in the context of the current frame, and view all of them simultaneously, you can create watches for them. The values of the expressions are updated with each step through the application, but are only visible when the application is suspended. Unlike the Expression Evaluation feature , these expressions are persisted as the part of your project.

This section describes how to add items to watches, change and remove watches.

Creating watches

To add an item to a watch

Do one of the following:

- In the Watches pane, click +, or just press `Insert`.
- On the Variables pane, in the Inspection window, or in the Evaluate Expression dialog box, right-click the desired item and choose Add to Watches on the context menu.
- Select the desired item in the Variables pane and drag it to the Watches pane.
- Select item in the editor, right-click it and select Add to Watches on the context menu.

Tip You can navigate from a backtrace in the Watches pane to the respective line of the source code. To do that, right-click a line of backtrace, and choose Jump to Source on the context menu, or just press `F4`.

Editing watches

To edit a watch

- To change the expression represented by a watch, right-click the desired watch and select Edit on the context menu.

Deleting watches

To remove a watch

1. In the Watches pane, select a watch to be deleted.
2. On the context menu, choose Remove Watch, or press `Alt+Delete`.

Inspecting Watched Items

On this page:

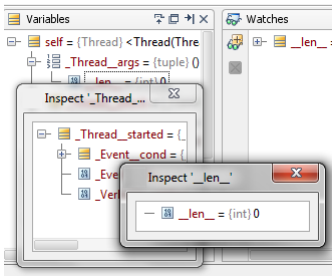
– Introduction

– Inspecting references

Introduction

IntelliJ IDEA helps inspect any variables or watches item in its own window. For example, if you need to examine several references in detail, you can open an inspection window for each of them. So doing, a separate window is created for each variable or watch reference and all of its child references.

The inspection windows are non-modal, and you can launch as many as of them required. All changes of the references are immediately reflected in the corresponding inspection windows.



Inspecting references

To inspect a reference

1. Select the item to be inspected on the Variables or Watches pane.

2. On the context menu, choose Inspect.

Setting Labels to Variables, Objects and Watches

On this page:

– Introduction

– Settings and removing labels

Introduction

You can add your own label to a variable, object, or watch and then reference it as if it were a local variable `<label-name>_DebugLabel` defined in the same context where the expression is evaluated. IntelliJ IDEA also displays these labels in suggestion pop-up lists for code completion in the Evaluate Expression dialog box.

Settings and removing labels

To set a label

1. Select the desired watch in the list
2. Select Mark Object on the context menu or press F11. The Select object label dialog box opens.
3. Specify the label name.
4. Click the Browse button ... to change the label color. Click OK when ready.

To remove a label, right-click the item and select Unmark Object on the context menu, or select the item in the list and press F11.

To navigate to the source code, do one of the following:

- Select the desired item in the Variables tab and press `F4`.
- Right-click an item in the Variables tab, and select Jump to Source from the context menu.

To navigate to an object's source

- Select an item in the Variables or Watches tab and press `Shift+F4`.
- Right-click an item in the Variables or Watches tab, and select Jump to Object Source from the context menu.

Customizing Views

On this page:

- Introduction
- Customizing Threads view
- Customizing Data view
- Custom type renderers
  - Rendering objects
  - Disabling custom type renderers
  - Switching between type renderers

Introduction

While exploring frames and their content, you might want to customize the way data is displayed. This section describes how to set such options.

Customizing Threads view

You can organize the way threads are shown in the list according to your needs.

To customize Threads view

1. Right-click anywhere in the Frames tab and choose Customize Threads View.
2. Specify viewing options. They are described in detail here .

Customizing Data view

You can organize the way data is shown in the Variables tab according to your needs.

To customize data view

1. Right-click anywhere in the Variables tab and select Customize Data Views.
2. Specify viewing options. They are described in detail here .

Custom type renderers

You can also specify your own type renderers instead of the default ones. They provide you the ability to customize how objects are displayed in the debugger, offering "logic-oriented" presentation of data vs. structure-oriented as it is by default.

Rendering objects

To render objects view

1. Open the Settings/Preferences dialog (`Ctrl+Alt+S`), click Debugger and select Type Renderers.
2. Click + to create a new renderer.
3. Specify the renderer name, the object class to which it applies, and which expressions to use while rendering. For details on rendering options refer to options description .
4. Define the appearance of the expanded node.
5. Click OK.

Disabling custom type renderers

To disable custom type renderer

1. Open the Settings dialog (`Ctrl+Alt+S`), click Debugger and select Type Renderers.
2. Select the type renderer to be disabled in the list and clear the check box next to its name in the list.
   Even if the type renderer is disabled, you can temporary switch to it while stepping through the program using the View as option.
3. Click OK.

Switching between type renderers

While stepping through the application in the Debug tool window, you can temporary switch between renderer schemes.

To switch between type renderers

1. Right-click the object instance in the Variables or Watches tab of the Debug tool window.
2. In the context menu, click View as, and then select the renderer from the list of the applicable type renderers.

Finding the Current Execution Point

When a program is suspended, the source file, associated with the current execution point, is opened in the editor. The current execution point (the next line to be executed) is marked with a blue line.

You can visit the other files, and then return to the current execution point using the actions described in this section.

To find the current execution point, do one of the following

- On the main menu, choose Run | Show Execution Point.
- Press `Alt+F10`.
- Click on the stepping toolbar of the Debug tool window.

Stepping Through the Program

In this section:

- Stepping Through the Program
  - Introduction
  - Stepping Through the Program
  - Tips and Tricks
- Choosing a Method to Step Into
- Improving Stepping Speed

Introduction

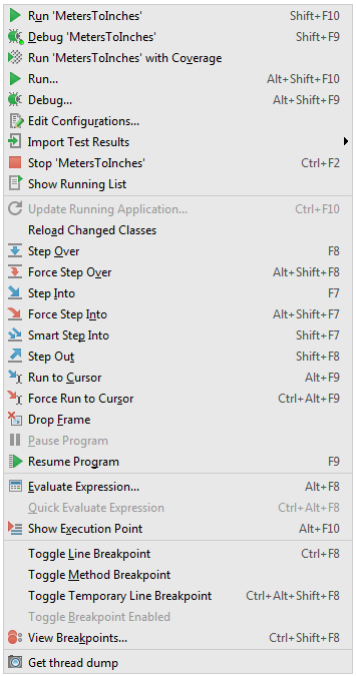
When a breakpoint is reached or your program is suspended , the Debug tool window becomes active and enables you to get control over the program's execution. For this purpose, you can use the Run menu commands, or the icons on the stepping toolbar of in the Debug tool window.

Each stepping action advances the execution point to the next execution location, depending on the action you choose.

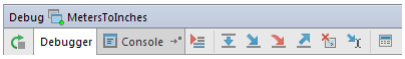
Stepping Through the Program

Do one of the following:

- On the main menu, choose Run | <stepping command>



- Use the keyboard shortcuts .
- Use the buttons in the stepping toolbar of the Debug tool window.



Tips and Tricks

- The Force Step Into command enables you to step into a method of a class not to be stepped into , for example, a standard Java SDK class .

  The classes, stepping into which is suppressed, are specified on the Stepping page of the Settings/Preferences dialog box.
- The Force Step Over command enables you to jump over the method call ignoring the breakpoints on the way.
- The Force Run to Cursor command enables you to jump to the cursor position ignoring existing breakpoints on the way.

Choosing a Method to Step Into

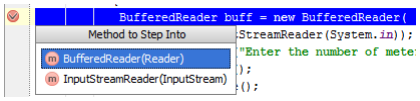
When you reach a line with calls of several methods, you can choose the method you want to step into.

Tip If you choose a method of a class stepping into which is suppressed on the Stepping page of the Settings dialog box, the suppression is overridden as when you invoke the Force Step Into command.

To choose a method to step into

1. On the main menu, choose Run | Smart Step Into or press `Shift+F7`.

2. In the pop-up window, choose the desired method from the list.



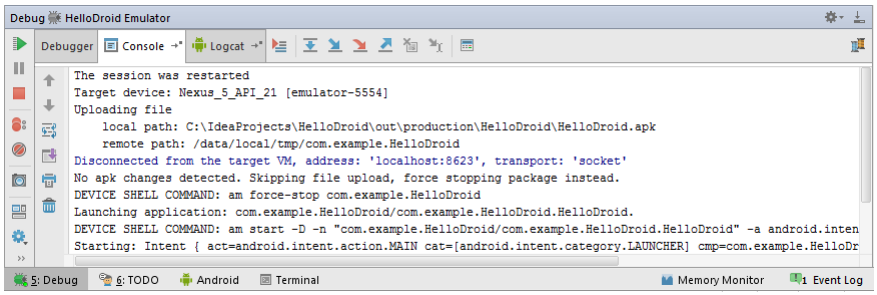
To improve stepping speed, follow these recommendations

- Try avoiding method and field breakpoints.
- If the Watch method return values option is enabled in Debug Toolbar | Settings, disable it.
- Turn off Alternate view for Collections classes by clearing the Enable alternative view for Collections classes check box at the Data Views page of the Debugger settings.
- Turn off the 'ToString' mode on the IDE level by clearing the Enable 'toString' object view check box on the Data Views page of the Debugger settings.
- Simplify the conditions for breakpoints and watchpoints, especially the frequently hit ones.
- Use filters (e.g., for class instances).
- During the debugging session, switch to a view with fewer elements.

Monitoring the Debug Information

Information of a debugging session is displayed in the dedicated tabs of the Debug tool window, named after the selected run/debug configuration.

For each session, use the Console tab to view the debugger messages and application output, and the Debug tab to monitor threads and frames.



When displaying and modifying local variables or watches values, IntelliJ IDEA uses the Default Encoding setting for the current project or the IDE encoding if no encoding is specified at the project level. The same setting is used when showing the PHP console script output.

Inline Debugging

On this page:

– Basics

– Enabling inline debugging

– Viewing inline debugging results

Basics

The inline debugging functionality facilitates the debugging procedure, as it lets you view the value of variables used in your source code right next to their usage, without having to switch to the Variables pane of the Debug tool window .

Enabling inline debugging

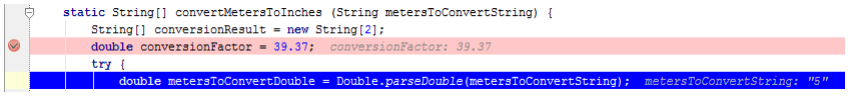
To enable the inline debugging functionality, do one of the following:

– In the Debug tool window toolbar, click the Settings icon and select the Show Values Inline option from the popup menu.

– Open the Data Views page of Setting/Preferences dialog, and select the check box Show values inline.

Viewing inline debugging results

If this option is enabled, when you launch a debug session and step through the program, the values of variables are displayed at the end of the lines where these variables are used.



Attaching to Local Process

On this page:

- Introduction
- Attaching to local process

Introduction

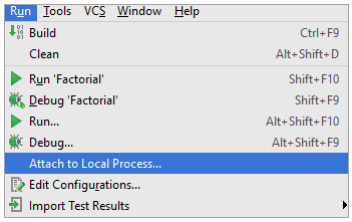
Attach to local process feature allows you to debug a project which you are developing in IntelliJ IDEA, but (for some reasons) are not able to launch directly from your IDE.

Attaching to local process

To attach to a local process, follow these general steps:

1. Launch the process intended for debugging. You can do it from operating system or using the IntelliJ IDEA terminal. Note, that your project shall exist in IntelliJ IDEA (as created or imported).

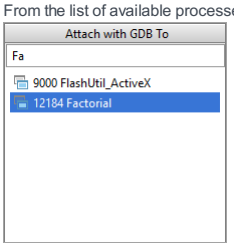
2. To find the process to attach to, do one of the following:
   - On the main menu, choose Run | Attach to Local Process.



   .

   - On the main menu, select the Help | Find Action option (or press `Ctrl+Shift+A`). In the list of actions that appears, find the desired action by typing the first letters, and select it.

3. From the list of available processes that appears, select the desired process. Simplify your search by typing the first letters of its name or PID



4. Proceed with debugging in the same way as you usually do it in IntelliJ IDEA (set breakpoints , step through the script , pause and resume the process, evaluate expressions etc.)

5. When finished, detach the process: select the Run | Stop menu option or click the Stop the process button of the Debug Tool Window .

Tutorial. Debugging with Chronon

On this page:

- What this tutorial is about
- Before you start...
- Preliminary steps
  - Preparing an example
  - Installing plugin
    - Changes to the UI
  - Creating run/debug configuration
    - Defining include/exclude patterns
- Running with Chronon
  - Opening an existing record
- What can you do with a record?
  - Switch between threads
  - Step through the application
  - Use bookmarks
  - Explore methods
  - Log values
  - Explore exceptions
- Summary

What this tutorial is about

This tutorial aims to walk you step-by-step through debugging a Java application with Chronon, a recorder and a "time-travelling" debugger.

Chronon records changes are made by your application while it is executing. The recordings are saved to files. You can later play these recordings back and share them among the team members.

Before you start...

First of all, it is essential to understand that Chronon is not literally a debugger - it only helps you record the execution progress and then play it back, like a videotape.

Second, make sure that:

- Your IntelliJ IDEA version is 13.1 or later. This tutorial is made for the version 14.1.
- The Chronon plugin is downloaded and installed .

Third, note that keyboard shortcuts on different keymaps may differ. This tutorial is made with the default keymap.

Preliminary steps

Preparing an example

Let's see how Chronon works on a simple example of a two-thread class. One thread performs quick sorting, while the second thread performs bubble sorting.

First, create a project as described in the page Tutorial: Creating, Running and Packaging Your First Java Application .

Next, create a package with the name `demo`, and, finally, add Java classes to this package. The first class is called `ChrononDemo.java` and it performs two-threaded array sorting:

```
package demo;

                    import org.jetbrains.annotations.NotNull;

                    import java.util.AbstractMap;
                    import java.util.Arrays;
                    import java.util.Random;

                    public class ChrononDemo {

                    public static final int SIZE = 1000;
                    public static final Random
                    GENERATOR = new Random();

                    public static void main(String[] args)
                    throws InterruptedException {
                    final int[] array =
                    new int[SIZE];
                    for (int i =
                    0; i < SIZE; i++) {
                    array[i] =
                    GENERATOR.nextInt();
                    }
                    final Thread quickSortThread =
                    new Thread(new Runnable() {
                    @Override
                    public void run() {
                    QuickSort.sort(Arrays.copyOf(array,
                    array.length));
                    }
                    }, "Quick sort");

                    final Thread bubbleThread = new Thread(new Runnable() {
                    @Override
                    public void run() {
                    BubbleSort.sort(Arrays.copyOf(array,
                    array.length));
                    }
                    }, "Bubble sort");

                    quickSortThread.start();
                    bubbleThread.start();

                    quickSortThread.join();
                    bubbleThread.join();
                    }
                    }
```

The second is the class `QuickSort.java` that performs quick sorting:

```
package demo;

                    class QuickSort {
                    private static int partition(int arr[], int
                    left, int right)
                    {
                    int i = left,
                    j = right;
                    int tmp;
                    int pivot = arr[(left + right) /
                    2];

                    while (i <= j) {
                    while (arr[i] < pivot)
                    i++;
                    while (arr[j] > pivot)
                    j--;
                    if (i <= j) {
                    tmp = arr[i];
                    arr[i] =
                    arr[j];
                    arr[j] =
                    tmp;
                    i++;
                    j--;
                    }
                    }

                    return i;
                    }

                    public static void sort(int arr[], int
                    left, int right)
                    {
                    int index = partition(arr, left, right);
                    if (left < index
                    - 1)
                    sort(arr, left, index - 1);
                    if (index < right)
                    sort(arr, index, right);
                    }

                    public static void sort(int arr[]) {
                    sort(arr, 0, arr.length -
                    1);
                    }
                    }
```

And, finally, the third one is the class `BubbleSort.java` that performs bubble sorting:

```
package demo;

                    public class BubbleSort {
                    public static void sort(int[] arr)
                    {
                    boolean swapped = true;
                    int j = 0;
                    int tmp;
                    while (swapped) {
                    swapped = false;
                    j++;
                    for (int i =
                    0; i < arr.length - j;
                    i++) {
                    if (arr[i] > arr[i
                    + 1]) {
                    tmp = arr[i];
                    arr[i] =
                    arr[i + 1];
                    arr[i + 1] = tmp;
                    swapped = true;
                    }
                    }
                    }
                    }
                    }
```

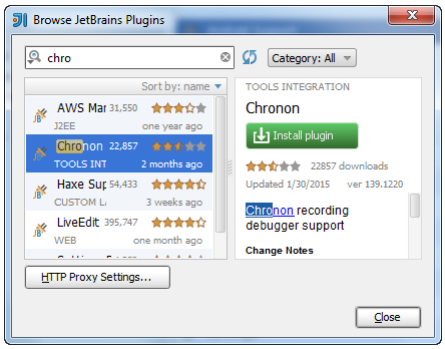
By the way, it is recommended to type the code manually, to see the magic IntelliJ IDEA's code completion in action.

Installing plugin

Open the Settings/Preferences dialog. To do that, click on the main toolbar, or press `Ctrl+Alt+S`.

Then click the page Plugins .

In this page, click the button Install JetBrains plugin... to download and install plugins from the JetBrains repository. In the Browse JetBrains Plugins dialog box, find the Chronon plugin - you can type the search string in the filter area:



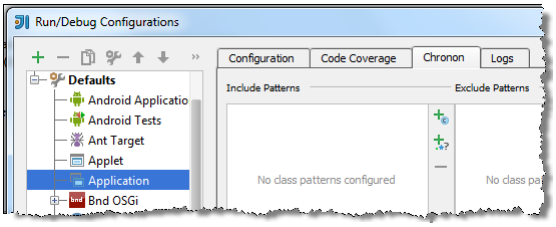
Install the plugin and restart IntelliJ IDEA for the changes to take effect.

Changes to the UI

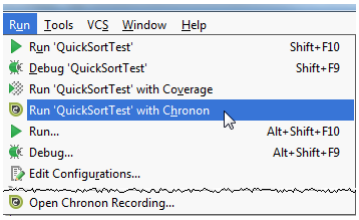
After restart, pay attention to the following changes:

- Dedicated Run with Chronon icon appears on the main toolbar. By now, this icon is disabled .

  It will become enabled as soon as the corresponding run/debug configuration appears.
- Chronon tool window (which becomes available on launching a run/debug configuration with Chronon, or on opening a Chronon record).
- Chronon tab appears in the run/debug configuration of the Application type (and some other types as well):



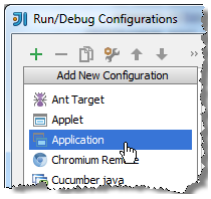
- Run menu is extended with two commands:
  - Run <run/debug configuration name> with Chronon
  - Open Chronon recording



Creating run/debug configuration

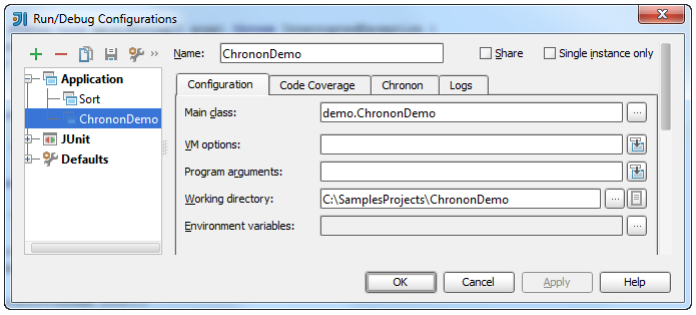
To launch our application, we need a run/debug configuration . Let's create one .

On the main menu, choose Run | Edit Configuration, and in the Run/Debug Configurations dialog box, click +. We are going to create a new run/debug configuration of the Application type, so select this type:



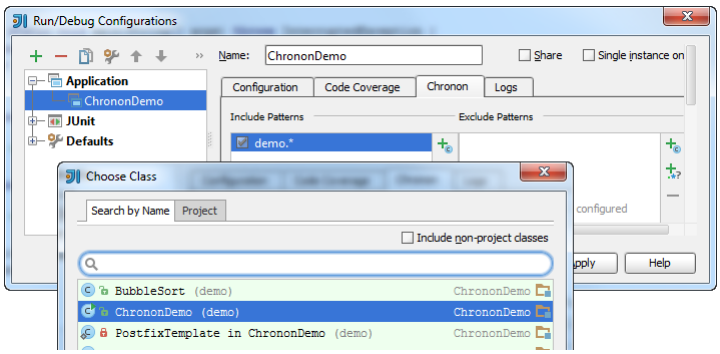
The new run/debug configuration based on the Application type appears. So far, it is unnamed and lacks reference to the class with the main method. Let's specify the missing information.

First, give this run/debug configuration a name. Let it be `ChrononDemo` . Next, press `Shift+Enter` and find the class named `ChrononDemo.java` that contains the main method. This class resides in the package demo:



Defining include/exclude patterns

Next, click Chronon tab. In this tab, you have to specify which classes IntelliJ IDEA should look at. This is done by Include / Exclude Patterns. In the Include Patterns area, click :

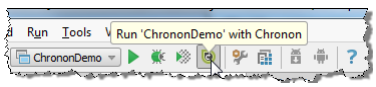


Now apply changes and close the dialog. The preliminary steps are done.

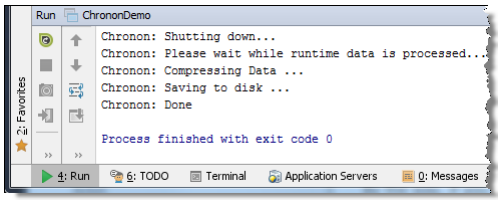
Running with Chronon

OK, it's time to launch our application. To do that, either click the Chronon button on the main toolbar, or choose Run | Run ChrononDemo with Chronon on the main menu.

Let's choose the first way:

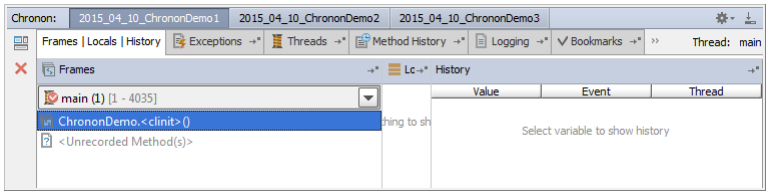


IntelliJ IDEA shows the progress bar, and then the Run tool window , where you can see the Chronon messages:



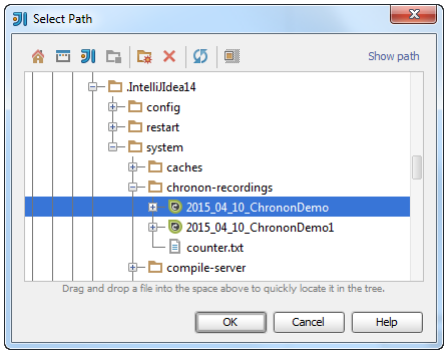
It is important to note that a Chronon record is NOT created, when you terminate your application by clicking . If it is necessary to stop an application and still have a Chronon record, click the Exit button on the toolbar of the Run tool window.

Then the Chronon tool window appears. It looks very much like the Debug tool window . In this tool window you see a record created by Chronon; so doing, each record shows in its own tab:



Opening an existing record

By the way, if you want to open one of the previous records, use Run| Open Chronon recording on the main menu, and then choose the desired record:

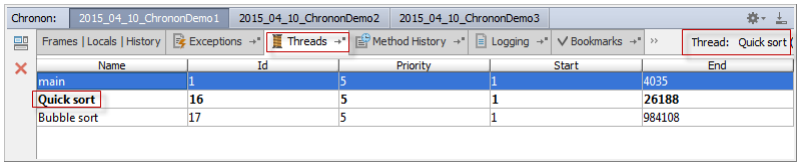


What can you do with a record?

In the Chronon tool window, you can:

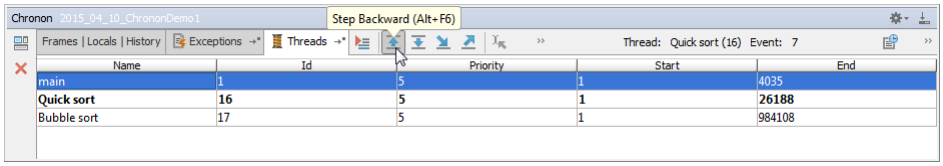
Switch between threads

This is most easy - just switch to the Threads tab, and double-click the thread you are interested in. The selected thread is shown in boldface in the list of threads; besides that, the information about the currently selected thread appears in the upper-right part of the Chronon tool window:



Step through the application

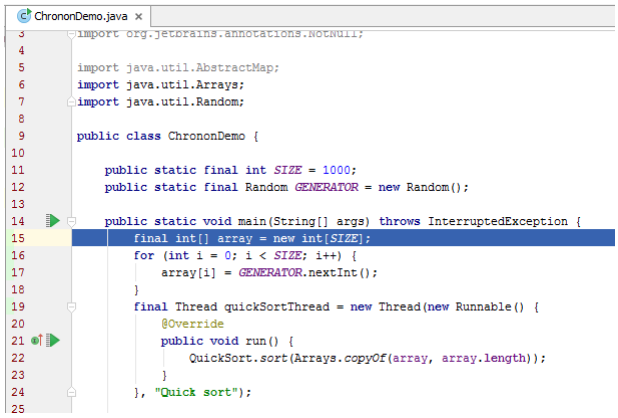
Actually, you can use either the stepping commands of the Run menu, or the stepping buttons of the Chronon tool window. Unlike the debugger that allows only stepping forward, Chronon makes it possible to step through the applications in the reverse direction also:



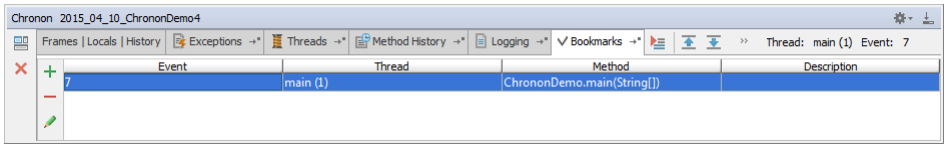
So, besides the traditional stepping buttons, there is Step Backwards button and Run Backwards to Cursor button .

Use bookmarks

Suppose you've stopped at a certain line, for example, in the main method of the class `ChrononDemo.java` :



You want to memorize this place with its event number, to be able to return to it from any other location. This is where the Bookmarks tab becomes helpful. Switch to this tab, and click . A bookmark for the current event, thread and method is created:

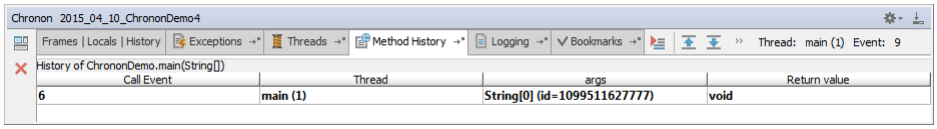


So doing, the bookmark memorizes the event number, thread and method name. Next, when you are in a different place of the code, clicking this bookmark in the Bookmarks tab will return you to this particular place.

Explore methods

If you look at the editor, you notice the icons in the left gutter next to the method declarations. Hovering the mouse pointer over such an icon shows a tooltip with the number of the recorded calls.

However, if you want to see a particular call event, then select the desired thread in the Threads tab (remember - the selected thread is shown in upper right part of the Chronon tool window), switch to the Method History tab, and track the execution history of a method:

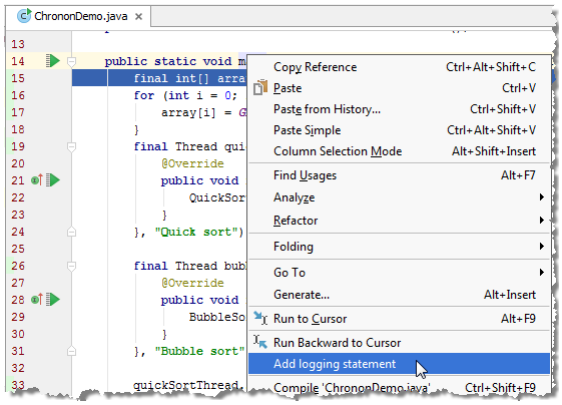


Thus you can explore the method execution, with the input and output data, which makes it easy to see how and where a method has been invoked.

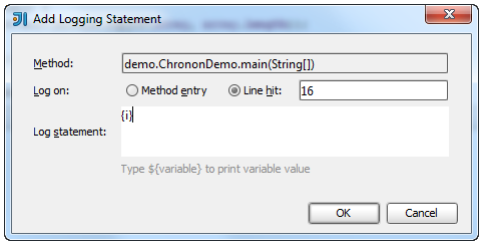
Log values

What is the Logging tab for? By default, it is empty. However, using this tab makes it possible to define custom logging statements and track their output during application run.

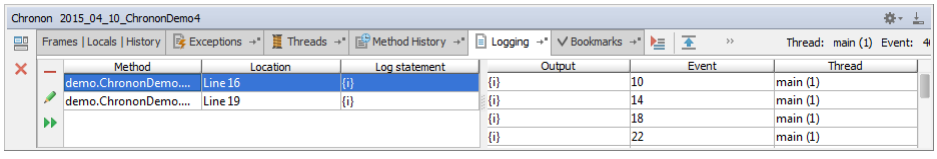
This is how logging works. In the editor, right-click the statement of interest, and choose Add logging statement on the context menu:



Then, in the dialog box that opens, specify the variable you want to watch in the following format:



Next, to make use of this logging statement, click in the toolbar of the Logging tab. In the right-hand side of the Logging tab, the output of the logged statement is displayed:



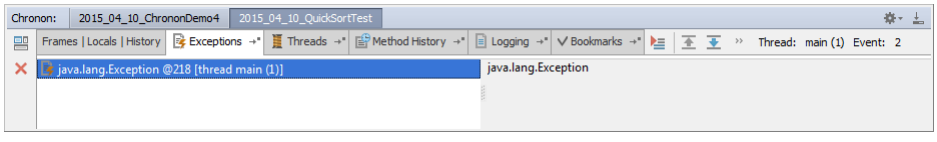
Note that such logging is equivalent to adding an output statement to your application; you could have added

```
System.out.println(<variable name>);
```

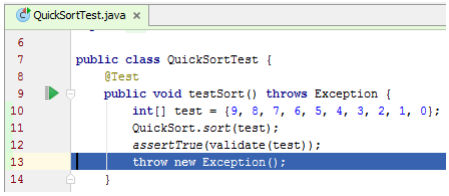
However, this would require application rebuild and rerun. With Chronon's logging facility, such complications are avoided.

Explore exceptions

Suppose you want to find out how and when an exception has occurred. The Exceptions tab shows all the exceptions that took place during an application execution:



If you double click an exception, IntelliJ IDEA will bring you directly to the place of the exception occurrence:



Summary

You've learned how to:

- Download and install Chronon plugin for IntelliJ IDEA
- Launch an application with Chronon recording.
- Open existing Chronon recordings.
- Work with the Chronon tool window. By the way, refer to the Hide/Restore Toolbar section of the Debug tool window reference to learn how to show/hide tabs.

This tutorial is over - congrats!

Testing

IntelliJ IDEA provides support for several language-specific testing frameworks . This section covers the issues that are common for all the supported testing frameworks:

- Configuring Testing Libraries
- Creating Tests
- Creating Test Methods
- Creating Run/Debug Configuration for Tests
- Performing Tests
- Monitoring and Managing Tests
- Viewing and Exploring Test Results
- Rerunning Tests
- Terminating Tests
- Tutorial: Test Driven Development

For the framework-specific testing procedures, refer to the sections:

- Testing Frameworks
- Testing ActionScript and Flex Applications
- Testing PHP Applications
- Unit Testing JavaScript

Configuring Testing Libraries

On this page:

– Introduction

– Adding test libraries

Introduction

The libraries for JUnit and TestNG are shipped with IntelliJ IDEA, but are not included in the classpath of your project or module by default. Consequently, when a test class is created, the references to the TestCase class or test annotations are not resolved.

To add the necessary library to the classpath, you can use the general procedure of configuring libraries . The corresponding libraries are located in the following directories:

– JUnit libraries (`junit.jar` and `junit-4.11.jar`): `<IntelliJ IDEA directory>\lib` .

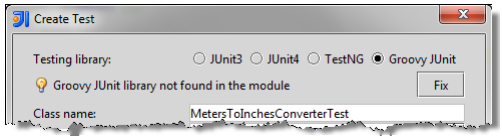
– TestNG library (`testng-jdk15.jar`): `<IntelliJ IDEA directory>\plugins\testng\lib` .

IntelliJ IDEA can add the necessary library to the classpath automatically. The corresponding features are available when creating a test for a class or when writing the code for a test .

Adding test libraries

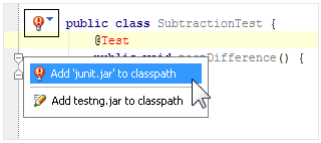
To add a test library to the classpath when creating a test for a class

1. In the editor, place the cursor within the line containing the class declaration.
2. Press `Alt+Enter` to view the available intention actions .
3. Select Create Test.
4. In the Create Test dialog, to the right of the text informing you that the corresponding library is not found, click Fix.



To add a test library to the classpath when writing the code for a test

1. In the source code of a test class, place the cursor within an unresolved reference to `TestCase` or annotation.
2. Press `Alt+Enter` to view the available intention actions .
3. Select Add <library> to classpath.



Creating Tests

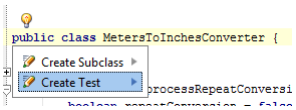
On this page:

- Creating a test class using the Create Test intention action
- Creating a test class using navigation command

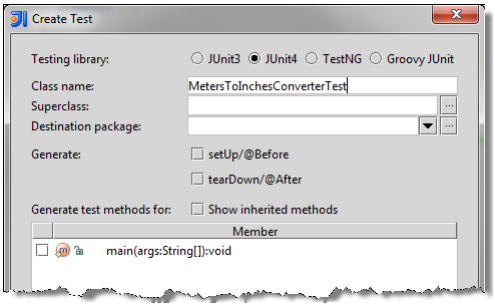
To create a test class using intention action

To create test cases for the supported testing frameworks, you can use the Create Test intention action . This intention action is available in the editor when the cursor is within the line containing the declaration of a class.

1. Open the class of interest in the editor.
2. Place the cursor within the line containing the class declaration.
3. Press `Alt+Enter` to show the list of intention actions.
4. Choose Create Test from the suggestion list.



5. In the Create Test dialog :
   1. Select the testing library to be used.
      If the selected library is missing in your module, click the Fix button. As a result, the corresponding library will be automatically added to the module libraries.
   2. Define the name and location of the test class to be generated.
      - In the Class name field, specify the name of the stub test class to be generated.
      - In the Superclass field, IntelliJ IDEA suggests the appropriate super class for JUnit3. For JUnit 4 and TestNG, this field is blank.
      - In the Destination package field, define where the generated test class should be placed.
   3. Specify whether you want the `setUp()` / `tearDown()` methods (for JUnit), or the `@Before` / `@After` annotations to be generated.
   4. In the table that shows the list of all the methods of the source class, select the ones you want to generate test methods for.
   5. Click OK.



To create a test class using navigation

1. Open the class of interest in the editor.
2. Place the cursor within the line containing the class declaration.
3. Do one of the following:
   - On the main menu, choose Navigate | Test.
   - On the context menu, choose Go to | Test.

   If the target test doesn't exist, you are prompted to create it. Refer to the section Navigating Between Test and Test Subject for details.

Creating Test Methods

To create stub test methods in JUnit test classes, you can use the IntelliJ IDEA code generation feature.

Note To create a test class with a complete set of test methods and fixtures, use the Create Test intention action.

Creating a stub test method in a JUnit test class

1. Open the corresponding JUnit test class in the editor.
2. Place the cursor where you want a new test method to be generated.
3. Press `Alt+Insert` and select Test Method from the Generate menu.

Creating Run/Debug Configuration for Tests

On this page:

- Introduction
- Creating run/debug configuration for tests

Introduction

You can run your tests (test cases, test suites, etc.) using run/debug configurations, in the way similar to running ordinary applications.

To create the necessary run/debug configurations, you can use the general procedure (see Creating and Editing Run/Debug Configuration) or various shortcuts available in IntelliJ IDEA. Using these shortcuts (implemented as context menu commands) you can create the run/debug configurations for:

- An individual test case.
- All tests in a directory or a package.
- Selected test classes which may be located in the same or different directories, packages or modules.
  This option is available only for JUnit.
- An individual test method.

Note You can run your tests even without creating the run/debug configurations for them. To do that, use the Run (`Ctrl+Shift+F10`) or the Debug command in corresponding context menus.

As a result, temporary run/debug configurations will be generated which you can then save.

Creating run/debug configuration for tests

To create a run/debug configuration for tests, follow these general steps:

1. To start creating the run/debug configuration, open the context menu for the item or items of interest and select the appropriate Create command. So, depending on what you want to create the run/debug configuration for, do one of the following:
   - If you want to create the run/debug configuration for an individual test, or all tests in a directory or a package:
     1. Right-click the corresponding test class, directory or package in the Project tool window .
     2. Select the Create ... command from the context menu.
        The command name depends on your current selection in the Project tool window and may look something like:
        - Create "All Tests" for a source or test directory.
        - Create "Tests in '<package_name>'" for a package.
        - Create "<test_name>" for a test class.
   - If you want to create the run/debug configuration for selected tests:
     1. Select the tests of interest in the Project tool window .
     2. Right-click one of the selected tests to open the context menu.
     3. In the context menu, select Create "<first_test_name> and <n> more".
   - If you want to create the run/debug configuration for a test method:
     1. Open the test class containing the method of interest in the editor.
     2. To open the context menu, right-click somewhere within the method code.
     3. In the context menu, select Create "<method_name>()".

   Note A similar context menu command is available for the whole test class in the editor. To open the context menu for the class, right-click somewhere outside the area occupied by the code of the methods (for example, in the line containing the class declaration).
2. In the dialog that opens, specify the run/debug configuration parameters and click OK. For more information, see the corresponding dialog description for JUnit or TestNG .

Tip To distinguish between the context menu commands for JUnit and TestNG, the following icons are used:
- for JUnit.
- for TestNG.

Performing Tests

On this page:

– Introduction

– Running or debugging a test

Introduction

Generally, IntelliJ IDEA runs and debugs tests in the same way as other applications, by running the run/debug configurations you have created . When doing so, it passes the specified test classes or methods to the test runner.

In many cases, you can initiate a testing session from a context menu. For this purpose, the Run and Debug commands are provided in certain context menus. For example, these commands are available for a test class, directory, or a package in the Project tool window . They are also available for a test class or method you are currently working on in the editor.

If you run a test for which there is no permanent run/debug configuration a temporary configuration is created. You can then save such a configuration using the Run/debug configuration dialog, if you want to reuse it later.

The tests run in the background, so you can execute several tests at the same time.

Each running configuration gets its own tab in the Run tool window (the Test Results tab). One tab can aggregate several tests.

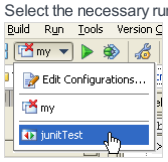
Running or debugging a test

To run or debug a test, follow these general steps:

To start running or debugging a test, you can use the main toolbar, or a context menu in the Project tool window or in the editor:

– Using the main toolbar:

1. Select the necessary run/debug configuration from the list on the main toolbar.



2. Click Run or Debug to the right of the list. (Alternatively, choose Run | Run (`Shift+F10`) or Run | Debug (`Shift+F9`) from the main menu.)

> **Tip** To see the list of available run or debug configurations and to quickly select the one you want, you can use the following keyboard shortcuts: `Shift+Alt+F10` for the run configurations, or `Shift+Alt+F9` for the debug configurations.

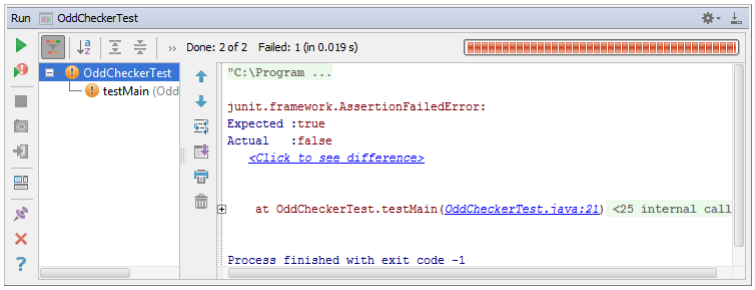
– Using a context menu:

Right-click a test class in the Project tool window, or open it in the editor, and right-click the background. On the context menu, choose Run < class name> or Debug....

> **Tip** For a test method, open the class in the editor and right click anywhere in the method. The context menu suggests the command Run / Debug <method name>.

Monitoring and Managing Tests

Test progress and results display in the dedicated test runner tabs of the Run tool window.



You can rerun , terminate, and suspend execution of tests same way as you do it for running applications. In addition to the common running actions, in the test runner you can:

- Navigate through the list of test cases using arrow keys.
- Navigate between failed tests using the and buttons or `Ctrl+Alt+Up` or `Ctrl+Alt+Down` keyboard shortcuts.
- View the total number of tests being run in the current session. The summary information is displayed in a message line on the top of the tool window. After completion of the tests, the message informs you about the number of failed tests and elapsed time.
- View testing progress in the progress bar.
- Show or hide information about the passed tests, using the button.
- Navigate from the stack trace to the problem location in the source code by clicking the hyperlink in the Output pane.
- Enable and disable the following functionality by clicking the cog button and selecting the relevant items from the context menu:
  - Monitoring execution of the current test.
  - Have the first failed test selected automatically upon completing the suit.
  - Navigating to the stack trace.
  - Automatic scrolling to the source code.
  - Have the corresponding source code opened at exceptions.
  - Have statistic shown in the Statistics pane.

Viewing and Exploring Test Results

On this page:

- Overview
- Viewing Statistics
- Important Note
- Viewing the Results of Previously Run Tests

Overview

Depending on the selected node in the tree view of tests, the Test Runner displays information on the various levels. In the Test Runner, you can view statistics of the tests, navigate to stacktrace, show or hide successful tests, and more.

Use the Testing toolbar to control visual representation of the test results.

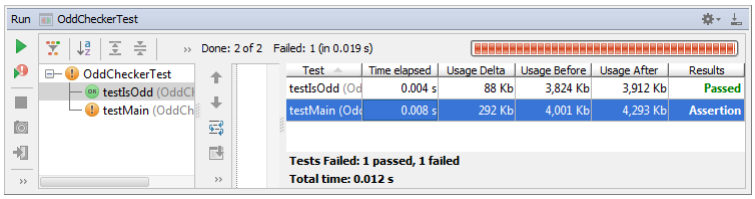
Viewing Statistics

The Statistics tab shows information about the elapsed time and memory usage of each test.

To view statistics:

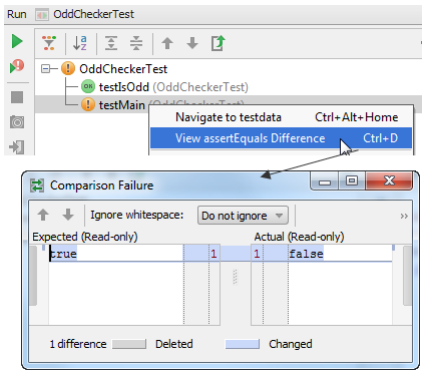
- Click on the Testing toolbar to reveal the drop-down menu, and then click the Show Statistics check command.

The values displayed in the Statistics tab are not accurate and only give an approximate estimate of the test performance. For example, if a garbage collector works during the test run, the memory usage shown in the Statistics tab is wrong.



Important Note

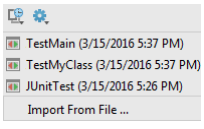
If a unit test contains string `assertEquals` failures, the test runner provides the ability to view differences between the compared strings. Choose the View assertEquals Difference on the context menu of the failed test that contains `assertEquals`, and explore differences in the Differences viewer:



Viewing the Results of Previously Run Tests

The results of a test session can be saved in two ways:

- Click to have the results of the selected test saved in a file. In the Export Test Results that opens, specify the file to save the output in and the format in which the data will be saved. If you want to view the test results later, choose the XML format.
- IntelliJ IDEA keeps track of test sessions and saves test results in its internal memory. Click to view the test results that you previously saved in an XML file or the results that IntelliJ IDEA has kept in its internal history. The pop-up menu that opens shows a list of internally saved results of test sessions, each item is supplied with the name of the run configuration and a time stamp:



- To view the results of a testing session from the IntelliJ IDEA history, select the item with the suitable run configuration and time stamp.
- To load the previously exported results, choose Import from file and then choose the required XML file in the dialog box that opens.

The loaded test results are shown in the tab and the name of the corresponding run configuration is displayed on the title bar. To re-run the tests from the loaded session, click .

Rerunning Tests

On this page:

– Introduction
– Rerunning a testing session
– Rerunning an individual test
– Rerunning a failed test
– Debugging a failed test

Introduction

You can repeat your test session, or individual tests without leaving your test runner tab of the Run tool window. The tests are performed again using the same run configuration as in the initial run.

Rerunning a testing session

Do one of the following:

– Click the rerun button on the toolbar of the Run tool window.
– Use `Ctrl+F5` keyboard shortcut.

Rerunning an individual test

1. In the testing tab of the test runner, right click a test case node or a test.
2. On the context menu, choose Run <test target>.

Rerunning a failed test

1. In the testing tab of the test runner, select a failed test.
2. In the Run toolbar , click Rerun Failed Tests .

Debugging a failed test

1. In the Test Runner tab , press `Shift` and click Rerun Failed Tests .
2. Select Debug from the Restart Failed Tests popup.

The tests that have failed will be rerun in the debug mode.

Terminating Tests

You can abort a running test session at any time. So doing, all tests that are current running will stop immediately. The icons of the tests in the test runner reflect the statuses of the tests (passed, failed, aborted, or never run).

To stop a testing session, do one of the following

- On the toolbar of the test runner, click the stop button .
- Use `Ctrl+F2` keyboard shortcut.

Test Driven Development

On this page:

- Introduction
- Prerequisites
- Creating Your First Test
- Writing the Test Body
- Running the Tests
- Implementing the Code
- Iterate
- Summary

Introduction

Whether you like to write your tests before writing production code, or like to create the tests afterwards, IntelliJ IDEA makes it easy to create and run unit tests. In this tutorial we're going to show how to use IntelliJ IDEA to write tests first (Test Driven Development or TDD).

Prerequisites

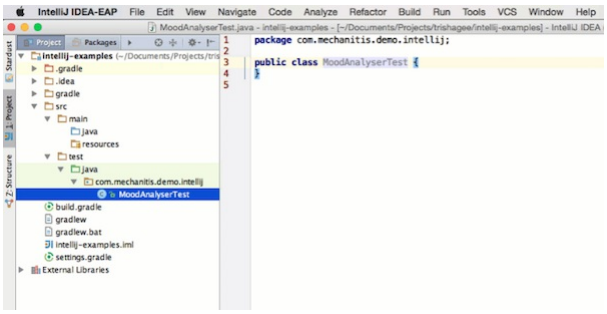
This tutorial assumes the following prerequisites:

- You have created a Java project in IntelliJ IDEA.
- You have folders for production and test code, either created manually or from when you created the project .
- You have JUnit 4 on the classpath, either by adding a test library , or adding a dependency in Gradle or Maven .
- Your source and test roots are correctly configured - source roots will appear as blue folders, test folders will have a green background.
- You have created the packages you want in your project.

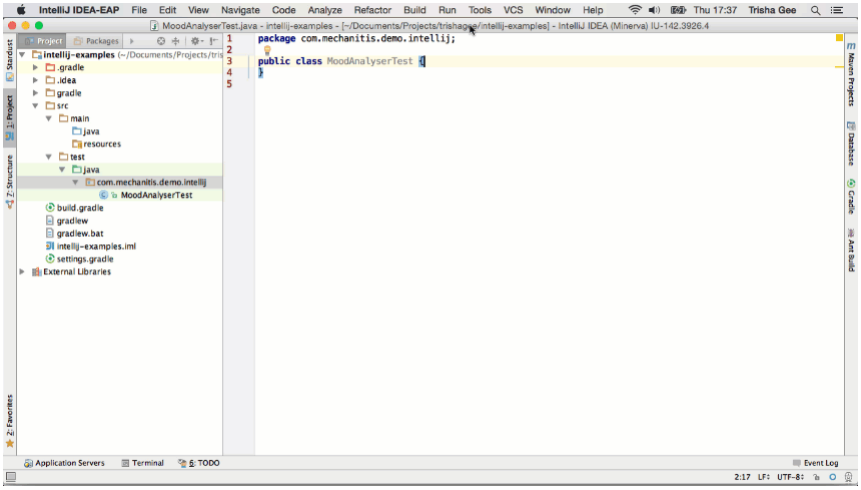
Creating Your First Test

Given that we're writing our tests first without necessarily having the classes we're testing available to us yet, we'll create our first test via the project panel.

1. Right click on the package you want to create the test in and select New | Java Class.
2. Enter the test name - given we're testing using JUnit, this will likely be [Something]Test, for example `MoodAnalyserTest` .



3. With the cursor placed somewhere inside the class's curly braces, press `Alt+Insert` .
4. Select Test Method | JUnit 4 from the menu. This will create a test method with the default template. Fill in a name for the test, press enter and the cursor will end up in the method body.

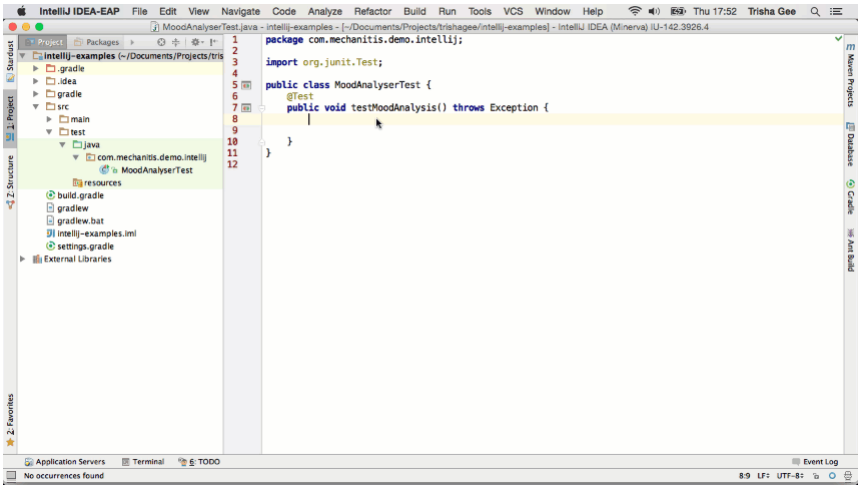


You can alter the default test method template - for example, if you wish to change the start of the method name from `test` to `should` .

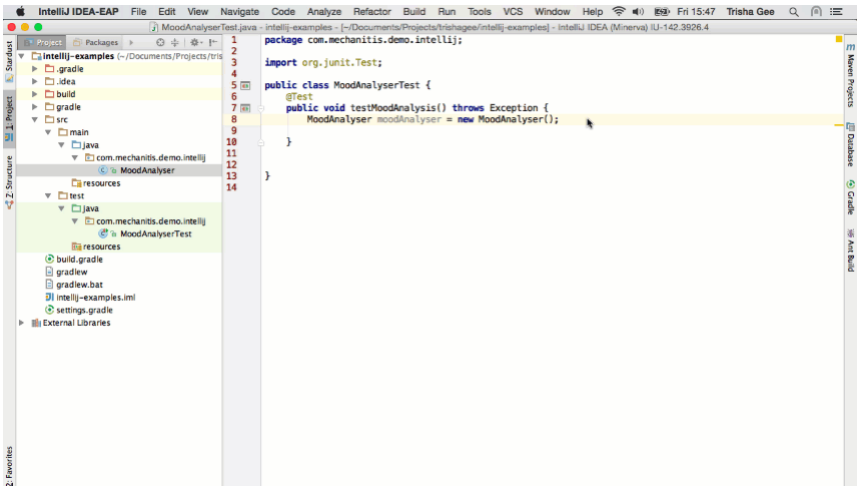
Writing the Test Body

It may seem counter-intuitive to write test code for classes and methods that don't exist, but IntelliJ IDEA makes this straightforward while keeping the compiler happy. IntelliJ IDEA can create classes and methods for you if they don't already exist.

1. Write your tests describing what you want to achieve, pressing `Alt+Enter` on any classes that don't exist and selecting "Create class '[ClassName]'". This will give you a minimum implementation that keeps the compiler happy.



2. Continue writing the test body, including names of methods that you need that don't exist, again you can use `Alt+Enter` and select "Create method '[methodName]'" to have IntelliJ IDEA create a bare skeleton method.



As always, you can use IntelliJ IDEA's refactoring tools to create variables to store results in, and IntelliJ IDEA will import the most appropriate classes for you, if the correct libraries are on the classpath.

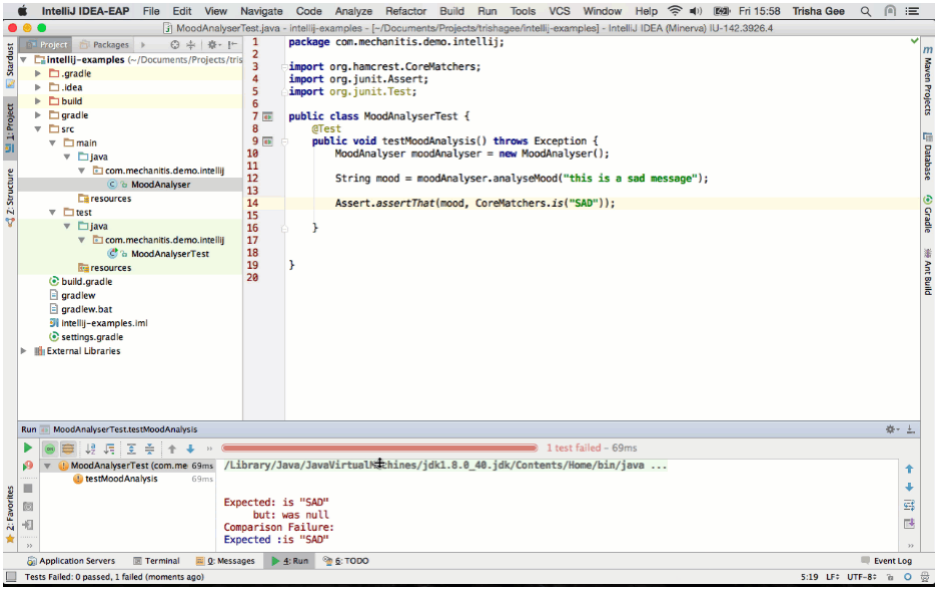
Running the Tests

When following a TDD approach, typically you go through a cycle of Red-Green-Refactor . You'll run a test, see it fail (go red), implement the simplest code to make the test pass (go green), and then refactor the code so your test stays green and your code is sufficiently clean.

The first step in this cycle is to run the test and see it fail.

Given that we've used IntelliJ IDEA's features to create the simplest empty implementation of the method we're testing, we do not expect our test to pass.

– From inside the test, press `Ctrl+Shift+F10` to run this individual test.
The results will be shown in the run dialog . The test name will have an icon next to it - either red for an exception, or yellow for an assertion that fails. For either type of failure, a message stating what went wrong is also shown.



Implementing the Code

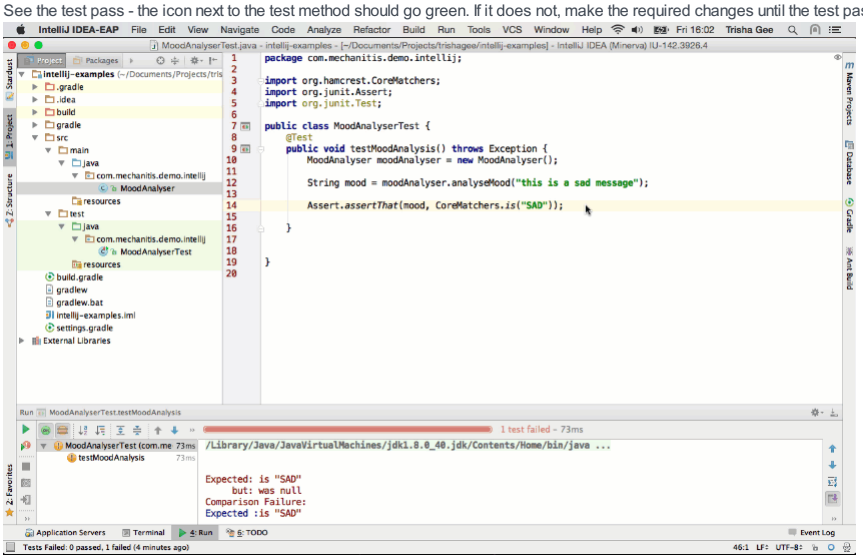
The next step is to make the tests pass, which means implementing the simplest thing that works.

1. You can navigate to the code being tested using the usual methods - clicking through on the method name, pressing `Ctrl+Alt+B` while the cursor is on the method name, or pressing `Ctrl+Shift+T` to switch between the test and the production code.

2. Make the change to the method to make the test pass. Often with TDD, the simplest thing that works might be hard-coding your expected value. We will see later how iterating over this process will lead to more realistic production code.

3. Re-run the test, using `Shift+F10` to re-run the last test.

4. See the test pass - the icon next to the test method should go green. If it does not, make the required changes until the test passes.



Iterate

Developing code is an iterative process. When following a TDD-style approach, this is even more true. In order to drive out more complex behaviour, we add tests for other cases.

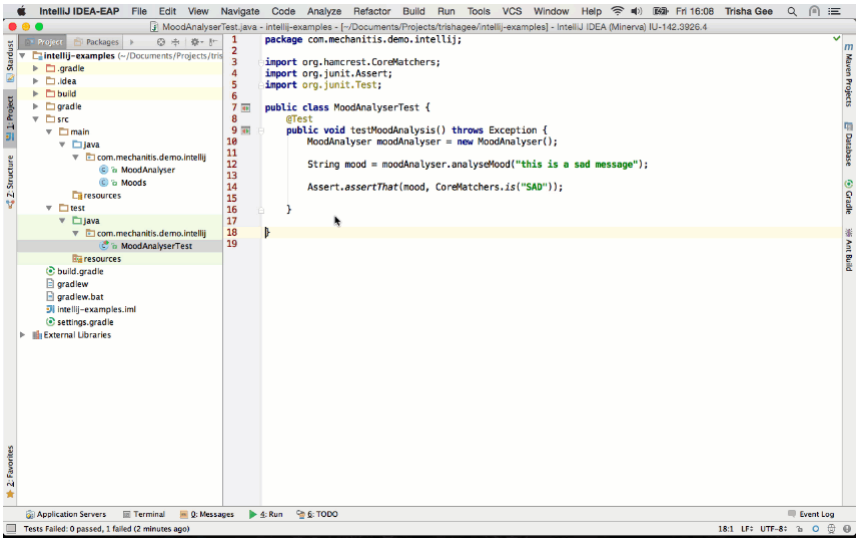
1. In your test class, use `Alt+Insert` again to create a new test method.

2. Pick a second test case that specifies more requirements of the method you're testing. Remember that you can use IntelliJ IDEA's features to create classes and methods to keep the compiler happy.

3. Run this second test case, showing that it fails for the correct reason.

4. Change the code in the method being tested to make this test pass.

5. Re-run both the tests by pressing `Ctrl+Shift+F10` inside the test class, not inside a single method, and see that both tests now pass. If either test fails, make the changes needed to the code to ensure the tests pass.



Summary

Writing your first test in a test-first style takes a small amount of setup - creating the test class, creating the test methods, and then creating empty implementations of the code that will eventually become production code. IntelliJ IDEA automates a lot of this initial setup.

As you iterate through the process, creating tests and then making the changes required to get those tests to pass, you build up a comprehensive suite of tests for your required functionality, and the simplest solution that will meet these requirements.

Working with artifacts

On this page:

- What is an artifact?
- Working with artifact configurations
- Building artifacts
- Running JAR artifacts
- Deploying artifacts to application servers and cloud platforms
- Examples

See also, Project Structure / Artifacts .

What is an artifact?

An artifact is an assembly of your project assets that you put together to test, deploy or distribute your software solution or its part. Examples are a collection of compiled Java classes or a Java application packaged in a Java archive, a Web application as a directory structure or a Web application archive, etc.

An artifact can be an archive file or a directory structure that includes the following structural elements:

- Compilation output for one or more of your modules
- Libraries included in module dependencies
- Collections of resources (web pages, images, descriptor files, etc.)
- Other artifacts
- Individual files, directories and archives.

Working with artifact configurations

Artifacts are generated according to artifact configurations. The artifact configurations are managed in the Project Structure dialog (File | Project Structure | Artifacts).

The key part of working with an artifact configuration is specifying the artifact structure and contents on the Output Layout tab .

Building artifacts

You can initiate building an artifact yourself: Build | Build Artifacts.

You can as well build an artifact by executing a run/debug configuration:

In the corresponding run/debug configuration, add the Build <ArtifactName> artifact task to the Before launch task list. The artifact will be built automatically when you execute the run/debug configuration (Run | Run or Run | Debug).

Including the Build artifact task in a run/debug configuration makes sense when the run/debug configuration somehow uses the corresponding artifact, e.g. starts the application packaged in a JAR artifact or deploys a WAR or EAR artifact to an application server.

By default, an artifact, when built, is placed into the `out/artifacts/<artifact_dir>` folder.

Running JAR artifacts

To run Java applications packaged in Java archives (JARs), IntelliJ IDEA provides the JAR Application run configurations. To create such a run configuration:

1. Open the Run/Debug Configurations dialog (e.g. Run | Edit Configurations).
2. Click + and select JAR Application.

Deploying artifacts to application servers and cloud platforms

Many of the artifact formats (e.g. WAR, Exploded WAR, EAR, Exploded EAR) are suitable for deployment to application servers and cloud platforms. Here is how you deploy such artifacts:

1. In a server or cloud run/debug configuration, specify the artifact to be deployed. (Use the Deployment tab or field.)
2. Execute the run/debug configuration or use the Deploy command ( ) in the Application Servers, Run or Debug tool window.

Examples

Examples of the procedures discussed on this page can be found in the following tutorials:

- Creating, Running and Packaging Your First Java Application
- Developing a Java EE Application

Deployment. Working with Web Servers

In this section:

- Deployment. Working with Web Servers
  - Basics
  - Interaction between IntelliJ IDEA and servers
- Configuring Synchronization with a Web Server
- Accessing Files on Web Servers
- Uploading and Downloading Files
- Comparing Deployed Files and Folders with Their Local Versions
- Editing Individual Files on Remote Hosts
- Running SSH Terminal
- Tutorial: Deployment in IntelliJ IDEA

Basics

Among numerous ways to configure your development and production environments the most frequent ones are as follows:

- The Web server is installed on your computer. The sources are under the server document root (for example, `/htdocs`), and you do your development right on the server.
- The Web server is installed on your computer but the sources are stored in another folder. You do your development, then copy the sources to the server.
- The Web server is on another computer (remote host). Files on the server are available through the FTP/SFTP/FTPS protocol, through a network share, or a mounted drive.

Now let's see how to use IntelliJ IDEA in the above environment configurations. IntelliJ IDEA assumes that all development, debugging, and testing is done on your computer and then the code is deployed to a production environment.

Please note the following:

- The reason to stick to this "local development - deployment" model lies in the way IntelliJ IDEA provides its coding assistance which includes code completion, code inspections & validations, code navigation, etc. All this functionality is based on the index of the project files which IntelliJ IDEA builds when the project is loaded and updates on the fly as you edit your code.
- To provide efficient coding assistance, IntelliJ IDEA needs to re-index code fast, which requires fast access to project files. The latter can be ensured only for local files, that is, files that are stored on you hard disk and are accessible through the file system. Therefore IntelliJ IDEA does not support the mode when you access your files over a network folder (very often it becomes slow and unresponsive, performs random look-ups for no obvious reason, etc).

Interaction between IntelliJ IDEA and servers

Interaction between IntelliJ IDEA and servers is controlled through server access configurations. Anytime you are going to use a server, you need to define a server access configurations, no matter whether your server is on a remote host or on your computer.

Taking into account all the above, let's define the following basic concepts related to synchronization between IntelliJ IDEA and servers.

- An in-place server is a server whose document root is the parent of the project root, either immediate or not. In other words, the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.
- A local server is a server that is running in a local or a mounted folder and whose document root is NOT the parent of the project root.
- A remote server is a server on another computer (remote host).
- The server configuration root is the highest folder in the file tree on the local or remote server accessible through the server configuration. For in-place servers, it is the project root.
- A local file/folder is any file or folder under the project root.
- A remote file/folder is any file or folder on the server, either local or remote.
  Suppose you have a project `C:/Projects/My_Project/` with a folder `C:/Projects/My_Project/My_Folder` and a local server with the document root in `C:/xampp/htdocs`. You upload the entire project tree to `C:/xampp/htdocs/My_Project`. In the terms of IntelliJ IDEA, the folder `C:/Projects/My_Project/My_Folder` is referred to as local and the folder `C:/xampp/htdocs/My_Project/My_Folder` is referred to as remote.
- Upload is copying data from the project TO the server, either local or remote.
- Download is copying data FROM the server to the project.

After you have configured synchronization with a server, you can upload, download, and manage files on it directly from IntelliJ IDEA. Moreover, you can suppress uploading or downloading specific files or entire folders. Finally, you can optimize you workflow by configuring content roots so specific folders are not involved in indexing, which significantly saves project indexing time.

Synchronization with servers, uploading, downloading, and managing files on them are provided via the Remote Hosts Access bundled plugin, which is by default enabled. If not, activate it in the Plugins page of the Settings dialog box. For details, see Enabling and Disabling Plugins .

Configuring Synchronization with a Web Server

In this section:

- Configuring Synchronization with a Web Server
  - Before you start
  - Basics
    - Server access configuration
- Creating an in-Place Server Configuration
- Creating a Local Server Configuration
- Creating a Remote Server Configuration
- Customizing Upload/Download
- Excluding Files and Folders from Upload/Download

Before you start

Synchronization with servers, uploading, downloading, and managing files on them are provided via the Remote Hosts Access bundled plugin, which is by default enabled. If not, activate it in the Plugins page of the Settings dialog box. For details, see Enabling and Disabling Plugins .

Basics

IntelliJ IDEA distinguishes among in-place, local, and remote servers, however the meaning of these terms in the context of IntelliJ IDEA slightly differs from their common meaning:

- An in-place server is a server whose document root is the parent of the project root, either immediate or not. In other words, the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.
- A local server is a server that is running in a local or a mounted folder and whose document root is NOT the parent of the project root.
- A remote server is a server on another computer (remote host).

For more information about possible configuration of the production and development environment and working with servers from IntelliJ IDEA, see Deployment. Working with Web Servers .

Server access configuration

IntelliJ IDEA controls interaction with Web servers through server access configurations. Anytime you are going to use a server, you need to define a server access configurations, no matter whether your server is on a remote host or on your machine.

A server access configuration defines:

- The server type (in-place, local, or remote).
- The computer (host) where the server is running. For in-place and local servers, IntelliJ IDEA presupposes that it is the current computer where your project is.
- The server access configuration root: the highest folder in the server hierarchy that can be accessed through the server configuration.
- The URL address to access the server configuration root. Both the HTTP and the HTTPS protocols are supported.
  To access a server through HTTPS, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized authority and import this certificate in the truststore/keystore of the Oracle JRE (Java Runtime Environment) on which IntelliJ IDEA runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

   ```
   <jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_
   ```

   If you are using the Oracle JRE bundled with IntelliJ IDEA, the default path to the truststore/keystore is `<%product_installation_folder>/jre/jre/lib/security/jssecacerts` or `<%product_installation_folder>/jre/jre/lib/security/cacerts` . Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts` .

2. When asked to enter a password for the truststore/keystore, specify the default one `changeit` .

3. Open the `IntelliJ IDEA.exe.vmoptions` file in the `<IntelliJ IDEA_installation_folder>/bin` and add the following line to it:

   ```
   -Djavax.net.ssl.keyStore=<path to keystore>
   ```

4. Restart IntelliJ IDEA.

   Learn more at Java6 and Java7 .

- The protocol to transfer the data through.
- Correspondence between local (project) folders, destination folders on the server, and URL addresses to access the data on the server. This correspondence is called mapping.

You can define as many configurations as necessary, thus enabling flexible switching between upload/download setups.

Creating an In-Place Server Configuration

On this page:

- Basics
- Creating a server configuration: specifying its name, type, and visibility
- Configuring access to an in-place server: specifying the URL address of the server document root
- Specifying the project root folder and the URL address to access it

Basics

An in-place server is a server whose document root is the parent of the project root, either immediate or not. In other words, the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.

To configure access to the server in this set-up, you only need to specify the URL address of the sever document root, appoint the project root folder, and specify the URL address to access it.

Creating a server configuration: specifying its name, type, and visibility

1. Open the Deployment page by doing one of the following:
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Deployment under Build, Execution, Deployment.
   - Choose Tools | Deployment | Configuration on the main menu.
2. In the left-hand pane, that shows a list of all the existing server configurations, click the Add toolbar button +. The Add Server dialog box opens.
3. Specify the server configuration name in the Name text box. From the Type drop-down list, choose the server configuration type In-place. Use the Up and Down keyboard keys to scroll through the list of server configuration types.
4. Use the Visible only for this project check box to configure the visibility of the server access configuration (deployment configuration).
   - Select the check box to restrict the use of the configuration to the current project. Such configurations cannot be reused outside the current project, they do not appear in the list of available configurations in other projects.
   - When the check box is cleared, the configuration is visible in all IntelliJ IDEA projects and the settings from, including SSH credentials, can be reused.
5. Click OK. The Add Server dialog box closes and you return to the Connection tab of the Deployment node.

Configuring access to an in-place server: specifying the URL address of the server document root

In the Web server root URL text box, type the URL address associated with the document root of your Web server as defined in the Web server configuration file. This URL address will be the starting point for building the URL address of your application. Both the HTTP and the HTTPS protocols are supported.

To access a server through HTTPS, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized authority and import this certificate in the truststore/keystore of the Oracle JRE (Java Runtime Environment) on which IntelliJ IDEA runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

```
<jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_na
```

   If you are using the Oracle JRE bundled with IntelliJ IDEA, the default path to the truststore/keystore is `<%product_installation_folder>/jre/jre/lib/security/jssecacerts` or `<%product_installation_folder>/jre/jre/lib/security/cacerts`. Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts`.

2. When asked to enter a password for the truststore/keystore, specify the default one `changeit`.
3. Open the `IntelliJ IDEA.exe.vmoptions` file in the `<IntelliJ IDEA_installation_folder>/bin` and add the following line to it:

```
-Djavax.net.ssl.keyStore=<path to keystore>
```

4. Restart IntelliJ IDEA.

Learn more at Java6 and Java7.

For example, the Apache httpd server configuration file is `httpd.conf`. The default document root is the `htdocs` folder and the default URL address to access the data in it is `http://localhost`. If you have changed the default port `80`, you have to specify the port explicitly: `http://localhost:<port>`.

Specifying the project root folder and the URL address to access it

1. Switch to the Mappings tab.
2. In the Local path text box, specify the full path to your project root folder. Type the path manually, or click the Browse button and choose the folder in the dialog box, that opens.
3. In the Web path on server text box, type the path to the project root folder relative to the server document root specified in the server configuration file. As you type, IntelliJ IDEA dynamically builds the URL address through which your project root folder will be accessible and shows it as a link in the Project URL read-only field. To check that the URL address is constructed correctly and ensures access to the project root, click the link.

Creating a Local Server Configuration

On this page:

- Basics
- Creating a server configuration: specifying its name, type, and visibility
- Specifying the server configuration root and the URL address to access it
- Example of specifying a server configuration root
- Mapping project folders with folders on the server and the URL addresses to access them
- Example of mapping project folders with folders on the server

Basics

A local server is a server that is running in a local or a mounted folder and whose document root is NOT the parent of the project root.

To configure access to the server in this set-up, you need to specify the following:

1. The server configuration root folder and the URL address to access it.
2. Correspondence between the project root folder, the folder on the server to copy the data from the project root folder to, and the URL address to access the copied data on the server. This correspondence is called mapping.

Creating a server configuration: specifying its name, type, and visibility

1. Open the Deployment page by doing one of the following:
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Deployment under Build, Execution, Deployment.
   - Choose Tools | Deployment | Configuration on the main menu.
2. In the left-hand pane, that shows a list of all the existing server configurations, click the Add toolbar button . The Add Server dialog box opens.
3. Specify the server configuration name in the Name text box. From the Type drop-down list, choose the server configuration type Local or mounted folder.
   When editing the server configuration name in the Name text box, use the Up and Down keys on your keyboard to change the preselected server access to type in the Type drop-down list.
4. Use the Visible only for this project check box to configure the visibility of the server access configuration (deployment configuration).
   - Select the check box to restrict the use of the configuration to the current project. Such configurations cannot be reused outside the current project, they do not appear in the list of available configurations in other projects.
   - When the check box is cleared, the configuration is visible in all IntelliJ IDEA projects and the settings from, including SSH credentials, can be reused.
5. Click OK. The Add Server dialog box closes and you return to the Connection tab of the Deployment node.
   Click the Use as Default toolbar button to have IntelliJ IDEA silently apply the current configuration in the following cases:
   - Automatic upload of changed files .
   - Manual upload of files without choosing the target host.
   - Comparing local files and folders with their remote versions.

Specifying the server configuration root and the URL address to access it

1. In the Folder text box of the Upload/download project files area, specify the server configuration root.
   The server configuration root is the highest folder in the file tree on the server that can be accessed through the server configuration. The easiest way is to use the document root of your Web server as defined in the Web server configuration file. However you can appoint any other existing folder under the document root.
2. In the Web server root URL text box of the Browse files on server area, specify the URL address of the server configuration root. This URL address will be the starting point for building the URL address of your application. Depending on your choice of the server configuration root, do one of the following:
   - Type the URL address associated with the document root of your Web server as defined in the Web server configuration file.
   - Type the URL address in the format `<URL of the server document root>/<path to the relevant folder relative to the server document root>`.

   Both the HTTP and the HTTPS protocols are supported.
   To access a server through HTTPS, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized authority and import this certificate in the truststore/keystore of the Oracle JRE (Java Runtime Environment) on which IntelliJ IDEA runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

   ```
   <jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_
   ```

   If you are using the Oracle JRE bundled with IntelliJ IDEA, the default path to the truststore/keystore is `<%product_installation_folder%>/jre/jre/lib/security/jssecacerts` or `<%product_installation_folder%>/jre/jre/lib/security/cacerts`.
   Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts`.
2. When asked to enter a password for the truststore/keystore, specify the default one `changeit`.
3. Open the `IntelliJ IDEA.exe.vmoptions` file in the `<IntelliJ IDEA_installation_folder>/bin` and add the following line to it:

   ```
   -Djavax.net.ssl.keyStore=<path to keystore>
   ```

4. Restart IntelliJ IDEA.

Learn more at Java6 and Java7 .

Example of specifying a server configuration root

For example, the Apache httpd server configuration file is `httpd.conf`, according to it, the default document root is the `htdocs` folder, and the default URL address to access the data in it is `http://localhost`. For the sake of simplicity, let's suppose that you are using the XAMPP package and it is installed in the root of the `C:/` drive.

So if you decide to copy your project files directly under the server document root, your server configuration root will be `C:\xampp\htdocs` and its URL will be `http://localhost:<port>`.

You can establish a more complicated folder structure on the server, for example, to have `MySite1` and `MySite2` folders under the server document root. In this case the you will have to decide which of these folders you will use in the current configuration, let it be `MySite2`. Accordingly, the server configuration root will be `C:\xampp\htdocs\MySite2` and its URL address will be `http://localhost:<port>\MySite2`.

Mapping project folders with folders on the server and the URL addresses to access them

Configure mappings, that is, set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . The easiest way is to map the entire project root folder to a folder on the server, whereupon the project folder structure will be repeated on the server, provided that you have selected the Create Empty directories check box in the Options dialog box . "For more details, see Customizing Upload/Download .

1. Switch to the Mappings tab.
2. In the Local Path text box, specify the full path to the desired folder in the project tree. In the simplest case it is the project root.
3. In the Deployment Path text box, specify the folder on the server where IntelliJ IDEA will upload the data from the folder specified in the Local Path text box. Type the path to the folder relative to the server configuration root. If the folder with the specified name does not exist yet, IntelliJ IDEA will create it, provided that you have selected the Create Empty directories check box in the Options dialog box . For more details, see Customizing Upload/Download .
4. In the Web Path text box, type the path to the folder on the server relative to the server configuration root. Actually, type the relative path you typed in the Deployment Path text box.

Example of mapping project folders with folders on the server

For example, if your project is `C:\My_Projects\Mapping_project`, the server document root is `C:\xampp\htdocs`, the server configuration root is `C:\xampp\htdocs\MySite2`, and its URL address is `http://localhost:<port>\MySite2`, fill in the fields as follows:

1. In the Local Path text box, type `C:\My_Projects\Mapping_project`.
2. In the Deployment Path text box, type `Mapping_project`.
3. In the Web Path text box, type `Mapping_project`.

Creating a Remote Server Configuration

On this page:

- Basics
- Creating a server configuration: specifying its name, type, and visibility
- Specifying user credentials defined during registration on the host
- Enabling connection to the server and specifying the server configuration root
- Mapping local folders to folders on the server and the URL addresses to access them
- Overloading the Deployment Destination by Configuring Nested Mappings

Basics

A remote server is a server on another computer (remote host).

To configure access to the server in this set-up, you need to specify the following:

1. Connection settings: server host, port, and user credentials.
2. The server configuration root folder and the URL address to access it.
3. Correspondence between the project root folder, the folder on the server to copy the data from the project root folder to, and the URL address to access the copied data on the server. This correspondence is called mapping.

Creating a server configuration: specifying its name, type, and visibility

1. Open the Deployment page by doing one of the following:
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Deployment under Build, Execution, Deployment.
   - Choose Tools | Deployment | Configuration on the main menu.
2. In the left-hand pane, that shows a list of all the existing server configurations, click the Add toolbar button +. The Add Server dialog box opens.
3. Specify the server configuration name in the Name text box. From the Type drop-down list, choose the server configuration type depending on the protocol you are going to use to exchange the data with the server.
   - FTP: choose this option to have IntelliJ IDEA access the server via the FTP file transfer protocol .
   - SFTP: choose this option to have IntelliJ IDEA access the server via the SFTP file transfer protocol.
   - FTPS: choose this option to have IntelliJ IDEA access the server via the FTP file transfer protocol over SSL (the FTPS extension).

   When editing the server configuration name in the Name text box, use the Up and Down keys on your keyboard to change the preselected server access to type in the Type drop-down list.
4. Use the Visible only for this project check box to configure the visibility of the server access configuration (deployment configuration).
   - Select the check box to restrict the use of the configuration to the current project. Such configurations cannot be reused outside the current project, they do not appear in the list of available configurations in other projects. For example, if this check box is selected in an SFTP configuration, you cannot use your SSH credentials from it when you configure a remote interpreter.
   - When the check box is cleared, the configuration is visible in all IntelliJ IDEA projects and the settings from, including SSH credentials, can be reused.

   See Configuring Node.js Interpreters and Configuring Remote PHP Interpreters for details.
5. Click OK. The Add Server dialog box closes and you return to the Connection tab of the Deployment dialog box.
   Click the Use as Default toolbar button to have IntelliJ IDEA silently apply the current configuration in the following cases:
   - Automatic upload of changed files .
   - Manual upload of files without choosing the target host.
   - Comparing local files and folders with their remote versions.

Specifying user credentials defined during registration on the host

1. Specify the registration mode:
   - To login in a regular mode, specify the login in the User name text box.
   - To enable anonymous access to the server with your email address as password, select the Login as anonymous check box.
2. Specify the way to authenticate to the server:
   - For FTP server, type your password and select the Save password check box to have IntelliJ IDEA remember it.
   - For SFTP server, choose the way to authenticate to the server. Do one of the following:
     - To use standard authentication, choose Password, specify the password, and select the Save password check box to have IntelliJ IDEA remember it.
     - To use SSH authentication via a key pair, choose Key pair (OpenSSH).
       To apply this authentication method, you need to have your private key on the client machine and your public key on the remote server you connect to. IntelliJ IDEA supports private keys generated using the OpenSSH utility. See http://wiki.qnap.com/wiki/How_To_Set_Up_Authorized_Keys for details.

       Type the private key file and the passphrase in the corresponding text boxes.
   - For FTPS server, specify your user name and password and choose the security mechanism to apply.
     - Choose Explicit to have the explicit (active) security applied. Immediately after establishing connection, the FTP client on your machine sends a command to the server to establish secure control connection through the default FTP port.
     - Choose Implicit to have the implicit (passive) security applied. In this case, security is provided automatically upon establishing connection to the server which appoints a separate port for secure connections.

Enabling connection to the server and specifying the server configuration root

1. Specify the host name of the FTP/SFTP/FTPS server to exchange data with and the port to which this server listens. The default values are:
   - 21 for FTP and FTPS
   - 22 for SFTP
2. In the Root path text box, specify the server configuration root relative to your user home which was defined when you registered your account. This folder will be the

highest one in the folder structure accessible through the current server configuration. Do one of the following:

- Accept the default value `/`, which points at the user home folder on the server.
- Type the path manually.
- Click the Browse button and select the desired folder in the Choose Root Path dialog box that opens.
- Click the Autodetect button and have IntelliJ IDEA detect the user home folder settings on the FTP/SFTP server and set up the root path according to them. The button is only enabled when you have specified your user name and password.

3. In the Web server root URL text box, type the URL address to access the server configuration root (The server configuration root is the highest folder in the file tree on the local or remote server accessible through the server configuration. For in-place servers, it is the project root.). Both the HTTP and the HTTPS protocols are supported. To access a server through HTTPS, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized authority and import this certificate in the truststore/keystore of the Oracle JRE (Java Runtime Environment) on which IntelliJ IDEA runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

```
<jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_
```

If you are using the Oracle JRE bundled with IntelliJ IDEA, the default path to the truststore/keystore is `<%product_installation_folder>/jre/jre/lib/security/jssecacerts` or `<%product_installation_folder>/jre/jre/lib/security/cacerts`. Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts`.

2. When asked to enter a password for the truststore/keystore, specify the default one `changeit`.
3. Open the `IntelliJ IDEA.exe.vmoptions` file in the `<IntelliJ IDEA_installation_folder>/bin` and add the following line to it:

```
-Djavax.net.ssl.keyStore=<path to keystore>
```

4. Restart IntelliJ IDEA.

Learn more at Java6 and Java7.

4. Click the Open button to make sure that the specified URL address is accessible and points at the correct Web page.

Mapping local folders to folders on the server and the URL addresses to access them

Configure mappings, that is, set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . The easiest way is to map the entire project root folder to a folder on the server, whereupon the project folder structure will be repeated on the server, provided that you have selected the Create Empty directories check box in the Options dialog box . "For more details, see Customizing Upload/Download .

1. Switch to the Mappings tab.
2. In the Local Path text box, specify the full path to the desired folder in the project tree. In the simplest case it is the project root.
3. In the Deployment Path text box, specify the folder on the server where IntelliJ IDEA will upload the data from the folder specified in the Local Path text box. Type the path to the folder relative to the server configuration root. If the folder with the specified name does not exist yet, IntelliJ IDEA will create it, provided that you have selected the Create Empty directories check box in the Options dialog box . For more details, see Customizing Upload/Download .
4. In the Web Path text box, type the path to the folder on the server relative to the server configuration root. Actually, type the relative path you typed in the Deployment Path text box.

Overloading the Deployment Destination by Configuring Nested Mappings

You can configure separate mappings for a specific folder under the your project root to have the contents of this folder synchronized with another location on the remote host.

Suppose you have configured the mappings as follows:

Local Path	Deployment Path
`<project_root>`	`ftp://.../htdocs/my_project`
`<project_root>/my_folder`	`ftp://.../htdocs/my_folder`

Then the files in your project will be uploaded as follows:

Local Path	Deployment Path
`<project_root>/file1.js`	`ftp://.../htdocs/my_project/file1.js`
`<project_root>/my_folder/file2.js`	`ftp://.../htdocs/my_folder/file2.js`

instead of `ftp://.../htdocs/my_project/my_folder/file2.js`

Customizing Upload/Download

On this page:

- Basics
- Setting common upload/download options
- Specifying additional protocol-specific customization options for FTP/SFTP/FTPS servers

Basics

In addition to the mandatory settings that ensure successful upload and download in various project - server set-ups, you can choose additional options to customize interaction with the server. Most of these options apply to all types of server access configuration. For FTP/FTPS/SFTP server configurations, you can specify additional protocol-specific options.

Setting common upload/download options

1. Open the Options dialog box by doing one of the following:
   - On the main menu, choose Tools | Deployment | Options.
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Build, Execution, Deployment node, and then click Options under Deployment.
2. In the Options dialog box that opens, specify additional settings:
   - To have IntelliJ IDEA skip specific files or entire folders during upload/download, in the Exclude items by name text box, specify the patterns that define the names of these files and folders. Use semicolons as delimiters. Wildcards are welcome.
     The exclusion is applied recursively. This means that if a matching folder has subfolders, the contents of these subfolders are not deployed either.

     For more details about excluding files and folders from upload/download, see Excluding Files and Folders from Upload/Download .
   - Specify the details of the upload/download procedure by selecting or clearing the corresponding check boxes.

Specifying additional protocol-specific customization options for FTP/SFTP/FTPS servers

1. Open the Deployment dialog box by doing one of the following:
   - Choose Tools | Deployment | Configuration on the main menu.
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Deployment under Build, Execution, Deployment.
2. In the Deployment dialog box, click the Advanced Options button and specify additional uploading settings in the Advanced Options dialog box that opens:
   - To set the client to the passive mode , select the Passive mode check box. In this mode, the client on your machine connects to the server to inform about being in the passive mode, receives the port number to listen to, and established data connection through the port with the received number. This mode is helpful when your machine is behind a firewall.
   - To have the hidden files and directories (those with names that start with a dot `.`) shown in the Server Browser Tool Window , select the Show Hidden Files check box.
   - To ensure compatibility in child file naming with your FTP server, select the Compatibility mode check box. This is helpful if the remote FTP server reports the following error:

     ```
     Invalid descendant file name <file name>
     ```

     Selecting this option may slow down synchronization with the server.

Excluding Files and Folders from Upload/Download

On this page:

- Basics
- Excluding a Folder on Server from Upload/Download after Project Creation
- Excluding a Local Folder from Upload/Download
- Excluding Files and Folders from Upload/Download by Name
- Removing the Exclusion Mark

Basics

Suppressing uploading, downloading, and synchronization for files or folders with sources ensures that the sources are protected against accidental update. When applied to non-sources, it saves system resources because huge amounts of media, caches, or temporal files are no longer copied hither and thither.

You may need to suppress upload/download in the following cases:

1. You are going to work with externally created and uploaded source code. To process these remote sources in IntelliJ IDEA, you have to download them and arrange them in a project. However, there are some sources that you should not update at all. On the other hand, the folders on the remote host also may contain huge amounts of media, caches, temporal files, that you actually do not need in your work.
2. You have already downloaded the data from the server and arranged them in a IntelliJ IDEA project. However, for this or that reason, you need to have some files or folders on the server protected against upload/download, for example, to prevent accidental overwriting.
3. The local copy of an application contains both source code and other data that you do not need to upload. Besides, you want to protect some sources against overwriting by mistake. In this case, you can suppress upload/download for all files and folders that should not be uploaded.

There are two ways to exclude folders from upload/download:

- Explicitly, by marking the corresponding paths as excluded in the Remote Host tool window or in the Excluded Paths tab of the Deployment dialog box .
  1. The names of all the not-excluded folders and files are displayed on green background. The names of excluded items are displayed without background.
  2. In the Remote Host tool window, you can exclude both entire folders and specific files.
- By name, that is, by specifying patterns that determine the names of files and folders to be excluded in the Exclude Items by Name text box of the Options dialog box.

Separate files can be protected against upload/download only through excluding them by name .

Note In either case, the exclusion is applied recursively. This means that if the path to a folder is explicitly marked as excluded or the folder name matches the pattern, the contents of all its subfolders, if any, are also protected against upload/download.

Excluding a Folder on Server from Upload/Download after Project Creation

1. Choose the required folder right in the Remote Host tool window:
   1. On the main menu, choose Tools | Deployment | Browse Remote Host or View | Tool Windows | Remote Host .
   2. In the Remote Host tool window that opens, select the relevant server configuration from the drop-down box.
   3. Select the folder to exclude and choose Exclude Path on the context menu of the selection.
2. Add the required folder to the list of excluded paths:
   1. Open the Deployment dialog box by doing one of the following:
      - Choose Tools | Deployment | Configuration on the main menu.
      - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Deployment under Build, Execution, Deployment.
   2. In the Deployment dialog box, switch to the Excluded Paths tab. The tab shows list of previously excluded local and remote folders.
   3. Click the Add deployment path button. An empty line is added to the list.
   4. Click the Browse button . The Select remote excluded path dialog box that opens shows the data on the host accessed through the selected server configuration. Select the required folder.
   5. When you OK, you return to the Excluded Paths tab, where the selected remote folder is added to the list.

Excluding a Local Folder from Upload/Download

1. Open the Deployment dialog box by doing one of the following:
   - Choose Tools | Deployment | Configuration on the main menu.
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Deployment under Build, Execution, Deployment.
2. In the Deployment dialog box, switch to the Excluded Paths tab. The tab shows list of previously excluded local and remote folders.
3. Click the Add local path button. In the empty line that is added to the list, specify the location of the folder to be protected against upload/download. Type the path manually or click the Browse button and choose the required folder in the dialog that opens .

Excluding Files and Folders from Upload/Download by Name

1. Open the Options dialog box by doing one of the following:
   - On the main menu, choose Tools | Deployment | Options.
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Build, Execution, Deployment node, and then click Options under Deployment.
2. In the Options dialog box that opens, specify the patterns that define the names of these files and folders in the Exclude items by name text box. Use semicolons as delimiters. Wildcards are welcome.
   The exclusion is applied recursively. This means that if a matching folder has subfolders, the contents of these subfolders are not deployed either.

Removing the Exclusion Mark

To return a file or folder to upload/download, select it and choose Remove Path from Excluded on the context menu of the selection. Returning a folder to upload/download

automatically affects all its subfolders and files.

Accessing Files on Web Servers

On this page:

- Basics
- Accessing a server
- Creating a folder on a server
- Removing a file or folder on a server
- Renaming a file or folder on a server
- Moving a file or folder

Basics

Once you have set up synchronization between your local application sources and the application sources on a server, you can create new folders, move, rename, and delete existing files and folders. You can also compare files and folders on the server with their local versions.

Please note, that for the sake of simplicity any file or folder under the project root is referred to as local and any file or folder on the server is referred to as remote, even if the server is installed on your machine. For details, see Deployment. Working with Web Servers .

Although IntelliJ IDEA supports direct editing of individual non-project files on servers, to keep your local and remote sources synchronized, configure automatic upload using the >Upload changed files automatically to the default server drop-down list in the Options dialog box.

Access to servers is controlled through server configurations of the type FTP, FTPS, SFTP, or Local or Mounted Folder.

Accessing a server

1. Open the Remote Host tool window. On the main menu, choose one of the following:
   - View | Tool Windows | Remote Host
   - Tools | Deployment | Browse Remote Host
2. Select the required server configuration from the drop-down list. The tool window shows a tree view of files and folders under the server configuration root. If no relevant server configuration is available in the drop-down list, click the Browse button ... and configure access to the server in the Deployment dialog box that opens.
   For details, see Configuring Synchronization with a Web Server .

Creating a folder on a server

Select the parent directory and choose Create Folder on the context menu.

Removing a file or folder on a server

Select the required file or folder and choose Delete on the context menu.

Renaming a file or folder on a server

Select the required file or folder in the tree view, choose Rename on the context menu, and specify the new name in the Rename dialog box that opens.

Moving a file or folder

1. Select the file or folder and choose Cut on the context menu.
2. Select the folder to move the file or folder to and choose Paste on the context menu. Then confirm the changes in the Move remote Items dialog box that opens.

   Alternatively, select the required file or folder and drag it to the new location. Then confirm the changes in the Move remote Items dialog box that opens.

Uploading and Downloading Files

On this page:

- Basics
- Uploading files and folders manually
- Uploading application sources automatically before application start
- Uploading changed files automatically
- Uploading checked-in files immediately after commit
- Suppressing automatic upload
- Downloading files

Basics

There are the following main ways to synchronize project files with their copies on the local or remote server:

- Manually , through a menu command.
- Automatically, every time a file is updated . In this case, the default server configuration is used.
  You can additionally configure this upload method to have only manually saved files uploaded.
- Automatically, before launching the application or starting a debugging session. This method is available for PHP applications only,
- Automatically after the changes to the files are checked in to your version control repository.

Besides uploading files, you can also have IntelliJ IDEA download files that were deployed according to the default server configuration .

Uploading files and folders manually

1. Choose Tools | Deployment on the main menu.
2. Do one of the following:
   - To run the upload according to the default configuration, choose Upload to <default server configuration>.
   - To run the upload according to a specific configuration, choose Upload to. Then from the pop-up menu, select the server configuration that defines access to the required server.

Uploading application sources automatically before application start

When working on a PHP application, you can have its sources automatically uploaded to a server every time you run the application or initiate a debugging session from IntelliJ IDEA.

1. During creation of a run/debug configuration , click the Add button + and choose the Upload Files to Remote Host task from the list.
2. In the Upload to Remote Host dialog box that opens, specify the following:
   - The server configuration to access the required server.
   - The local root folders to upload. All the folders and files under the specified roots will be uploaded.

Uploading changed files automatically

IntelliJ IDEA considers a local file changed as soon as it is saved either automatically or manually. IntelliJ IDEA performs automatic save on a regular scheduled basis. Manual save is invoked by choosing File | Save All on the main menu or pressing `Ctrl+S` .

1. Open the Options dialog box by doing one of the following:
   - On the main menu, choose Tools | Deployment | Options.
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Build, Execution, Deployment node, and then click Options under Deployment.
2. In the Upload changed files automatically to the default server drop-down list, specify when you want IntelliJ IDEA to upload changed files:
   - To have IntelliJ IDEA upload any saved file no matter whether the save was invoked manually or automatically, choose Always.
   - To have IntelliJ IDEA upload only files that were saved manually, choose On explicit save action.

Uploading checked-in files immediately after commit

1. Start checking in your changes .
2. In the After Commit area, choose the target server from the Upload file to drop-down list. Choose one of the existing server access configurations or create a new one: click the Browse button ... and configure access to the relevant server in the Deployment dialog box that opens.
3. To have IntelliJ IDEA automatically upload checked in files to the chosen server, select the Always use selected server check box.

Suppressing automatic upload

1. Open the Options dialog box by doing one of the following:
   - On the main menu, choose Tools | Deployment | Options.
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Build, Execution, Deployment node, and then click Options under Deployment.
2. From the Upload changed files automatically to the default server drop-down list, choose Never.

Downloading files

Choose Tools | Deployment | Download from <default server configuration> on the main menu. Note that this functionality is available only if you have previously appointed a configuration as default.

Comparing Deployed Files and Folders with Their Local Versions

On this page:

- Basics
- Opening the Remote Host tool window
- Comparing a remote folder with its local version
- Comparing a remote file with its local version
- Comparing a local folder with its remote version
- Comparing a local file with its version on a server
- Comparing two folders in the Difference Viewer
- Synchronizing contents of folders

Basics

Correspondence between files and folders under the project root and their copies on a server is set through server access configuration mappings . Please note, that for the sake of simplicity any file or folder under the project root is referred to as local and any file or folder on the server is referred to as remote, even if the server is installed on your machine. For details, see Deployment. Working with Web Servers . For details, see Configuring Synchronization with a Web Server .

It is clear, that a local file or folder can be mapped to an unlimited number of remote counterparts, while each remote file or folder is mapped to one and only one file or folder respectively.

Consequently, for each remote file or folder, IntelliJ IDEA unambiguously detects its local version to compare with, so you can have remote items compared with their local counterparts at any time. On the contrary, remote versions of local files or folders cannot be identified uniquely, therefore IntelliJ IDEA can compare local items only with their remote versions mapped through the default server access configuration .

IntelliJ IDEA provides a dedicated Differences Viewer for Folders for comparing files in remote folders and their local versions against the file size, content, or timestamp. The Differences Viewer shows the contents of the local and remote directories in the Item List, in the left and right panes respectively. The contents of the selected file are shown in the lower pane, with the differences being color-highlighted.

The remote files in the Difference Viewer have the status `read-only` . This means that you cannot update them directly in the Difference Viewer. Make all the necessary changes to the local version of the file in question and upload the updated file to the server.

Besides exploring differences, the tool also provides interface for synchronizing the contents of folders.

Opening the Remote Host tool window

Do one of the following:

- On the main menu, choose Tools | Deployment | Browse Remote Host.
- On the main menu, choose View | Tool Windows | Remote Host. Note that the tool window can be accessed this way after you have opened it using the Tools | Deployment | Browse Remote Host.

Comparing a remote folder with its local version

1. Open the Remote Host tool window .
2. Select the folder in question. Then choose Tools | Deployment | Sync with Local on the main menu or Sync with local on the context menu of the selection.
3. In the Differences Viewer for Folders that opens, explore the differences and synchronize the files , where applicable.

Comparing a remote file with its local version

1. Open the Remote Host tool window .
2. Select the file in question, and then choose Compare with local version on the context menu of the selection.
3. In the Differences Viewer for Files dialog box, that opens, explore the differences and apply them, if necessary, using the » and « buttons. For details, see Viewing Differences Between Files .

Comparing a local folder with its remote version

This action is available only if you have a default server access configuration appointed.

1. Select the folder in question in the Project tool window, and then choose Tools | Deployment | Sync with Deployed to <default server access configuration> on the main menu.
2. In the Differences Viewer for Folders that opens, explore the differences and synchronize the files , where applicable.

Comparing a local file with its version on a server

This action is available only if you have a default server access configuration appointed.

1. Select the file in question in the Project tool window, and then choose Tools | Deployment | Compare with Deployed Version on <default server access configuration> on the main menu.

   Note When no default server access configuration is appointed, this menu item is disabled and its name is changed to Compare Local File with Deployed Version.

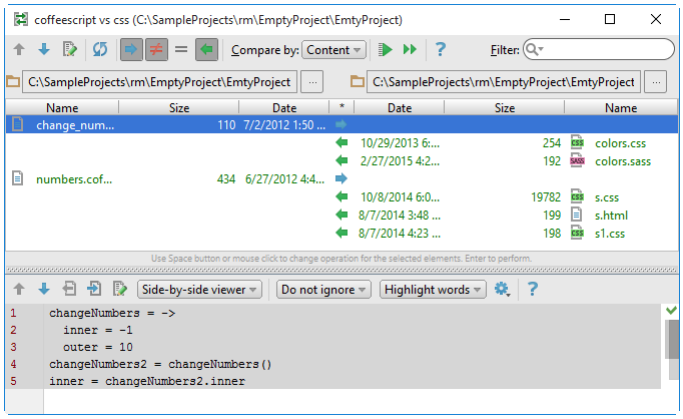
2. In the Differences Viewer for Files dialog box, that opens, explore the differences and apply them, if necessary, using the » and « buttons. For details, see Viewing Differences Between Files .

Comparing two folders in the Difference Viewer

Configure the layout of the Items List. Use the toolbar buttons to narrow down or widen the set of items to show. For example, show or hide files that are present only locally or remotely, equal files, different files, files excluded from synchronization , etc.

Specify the parameter for comparison. In the Compare by drop-down list, select one of the possible options (contents, size, or time stamp). Filter the folders' contents. To do that, type filtering string in the Filter text field, and press `Enter` to apply it. Using the asterisk `*` wildcard to represent any number of characters is welcome. To switch to another pair of folders to compare, update the fully qualified paths to them. Click the Browse button … next to the Paths read-only fields and choose the required folders in

the dialog box that opens . Explore the detected differences between files in the Differences Pane.



Synchronizing contents of folders

The contents of the remote folder are always shown in the right-hand pane of the Items List, the contents of its local version are always shown in the left-hand pane.

1. For each pair of items, in the * field specifies the action to apply. Click the icon in the field until the required action is set.
   - the file will be uploaded, possibly overwriting the remote version.
   - the file will be downloaded, possibly overwriting the local version.
   - the files are treated identical with regard to the selected criterion of comparison. No action will be performed by default.
   - the files differ with regard to the selected criterion of comparison. No action will be performed by default. Explore the differences in the Differences Pane of the Difference Viewer and change the intended action by clicking the icon.
     The remote files in the Difference Viewer have the status `read-only` . This means that you cannot update them directly in the Difference Viewer. Make all the necessary changes to the local version of the file in question and upload the updated file to the server.
   - the file is present only locally or remotely and will be removed.
2. Do one of the following:
   - To synchronize the currently selected item, click the Synchronize Selected button on the toolbar.
   - To synchronize all the items, click the Synchronize All button on the toolbar.

Editing Individual Files on Remote Hosts

On this page:

- Introduction
- Editing files on remote hosts

Introduction

Once you've set up synchronization with a remote host, you can open individual files directly from the remote host and edit them in IntelliJ IDEA, without adding/downloading them to the local project. Note that for such files not included in a project, some IntelliJ IDEA features are not supported.

To take advantage of debugging, refactorings, and other advanced features, consider including the files into a project; see Accessing Files on Web Servers for details.

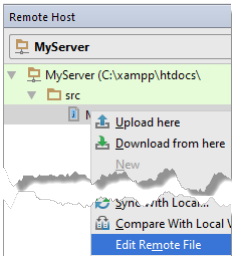
Editing files on remote hosts

To edit a file on a remote host

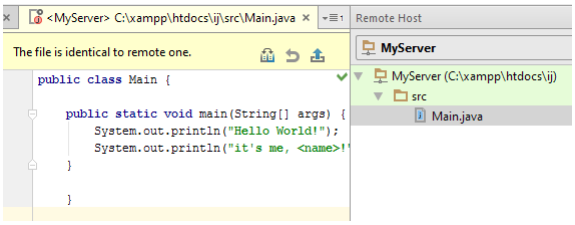
1. Open the Remote Host tool window. On the main menu, choose one of the following:
   - View | Tool Windows | Remote Host
   - Tools | Deployment | Browse Remote Host

2. Select the required server configuration from the drop-down list. The tool window shows a tree view of files and folders under the server configuration root. If no relevant server configuration is available in the drop-down list, click the Browse button and configure access to the server in the Deployment dialog box that opens.
   For details, see Configuring Synchronization with a Web Server .

3. To start editing a file, double click its name or select the file name in the Remote Host Tool Window , and choose Edit Remote File on the context menu:

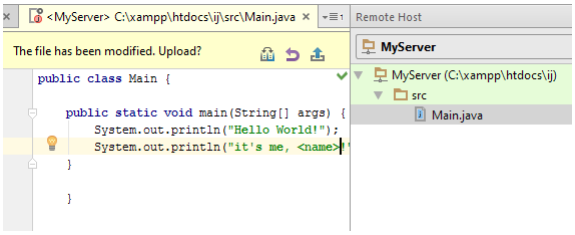


   The file opens in the IntelliJ IDEA editor, without being added or downloaded to the local project.



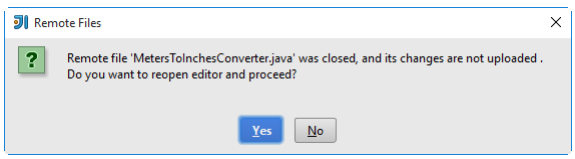
   When you work with a remote file, a special toolbar appears at the top of the editor, showing the editing status ("The file is identical to the remote one" or "The file has been changed. Upload?".

   Remote files can be easily distinguished from local ones by looking at the annotation, which includes the server name (in our case it is <MyServer>) .



4. When you are done editing the file, do one of the following:
   - To upload the file to the remote host, click .
   - To compare the currently opened file with the last uploaded version, click . The files are opened in the standard IntelliJ IDEA Difference Viewer, see Differences viewer for files .
   - To discard the changes introduced to the file after the last upload, click .

As it has been mentioned above, an individual file is not added to a project. As a result, all the changes to it are discarded as soon as you close the file or the currently opened project unless these changes have been uploaded. To prevent losing your data accidentally, IntelliJ IDEA displays the following dialog box when you attempt to close the edited file or the entire project:



Running SSH Terminal

On this page:

- Introduction
- Preparing to work in the SSH Terminal
- Launching an SSH Terminal

Introduction

You can launch an SSH Session right from IntelliJ IDEA. By running commands in a dedicated SSH terminal, you can access data on a remote Web server or a Vagrant instance (virtual machine) via an SSH tunnel, mainly upload and download files.

Preparing to work in the SSH Terminal

1. Make sure the SSH Remote Run plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .
2. Make sure, an SSH server is available in the destination environment: a remote Web server or a Vagrant instance (virtual machine).
3. Register an account on the SSH server in the destination environment and generate a pair od SSH keys or a password, depending on the server policy.
4. Appoint the destination environment and specify the settings to establish connection with it:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click SSH Terminal under Tools. The SSH Terminal page opens.
   2. In the Connection settings area, appoint the destination environment:
      - Current Vagrant: select this option to have the commands in the SSH Terminal executed on the currently running Vagrant virtual machine. For details, see Vagrant .
      - Deployment server: select this option to have the commands in the SSH Terminal executed on the local or remote Web server accessible through one of the server access configurations . From the drop-down list, choose the server access configuration that specifies the destination environment and the settings to establish connection to it.
        - Select server on every run: if this option is selected, you will have to choose the desired server access configuration from the pop-up window, every time you choose Tools | Start SSH Session on the main menu.
        - If the desired server access configuration does not appear in the drop-down list, click the link Configure Servers, and define one in the Deployment page. For details, see Configuring Synchronization with a Web Server .
   3. From the Default encoding drop-down list, select the desired encoding to be used in the SSH terminal.

Launching an SSH Terminal

1. On the main menu, choose Tools | Start SSH Session.
2. Depending on the connection settings, defined in the SSH Terminal page of the Settings dialog, the following types of behavior are possible:

Tutorial: Deployment in IntelliJ IDEA

On this page:

- What this tutorial is about
- Before you start
- Configuring a deployment server
  - What is specified in the Connection tab?
  - What is specified in the Mapping tab?
- Browsing remote hosts
- Deployment tools
  - Uploading
  - Comparing remote and local versions
  - Downloading
- Synchronizing changes
- Automatic upload to the default server
  - Defining a server as default
  - Enabling automatic upload
- Uploading external changes
- Summary

What this tutorial is about

This tutorial aims to take you step-by-step through configuring an managing deployment of your code to remote hosts, using IntelliJ IDEA.

Before you start

Make sure that:

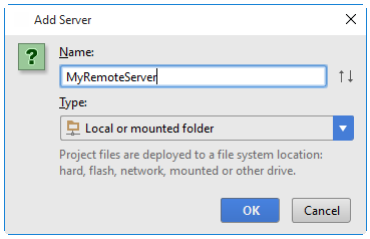
- You are working with IntelliJ IDEA version 15.0 or higher. This tutorial is prepared with IntelliJ IDEA 2016.1.
- You have access right to a remote host you want your code to be deployed on.

Also note that this tutorial is created on Windows 10 and makes use of the default keyboard shortcuts scheme. If you are working on a different platform, or use another keyboard scheme, the keyboard shortcuts will be different.

Configuring a deployment server

On the main toolbar, click to open the Settings/Preferences dialog, and choose the page Deployment (actually, you can access the same page by choosing Tools | Deployment | Configuration on the main menu).

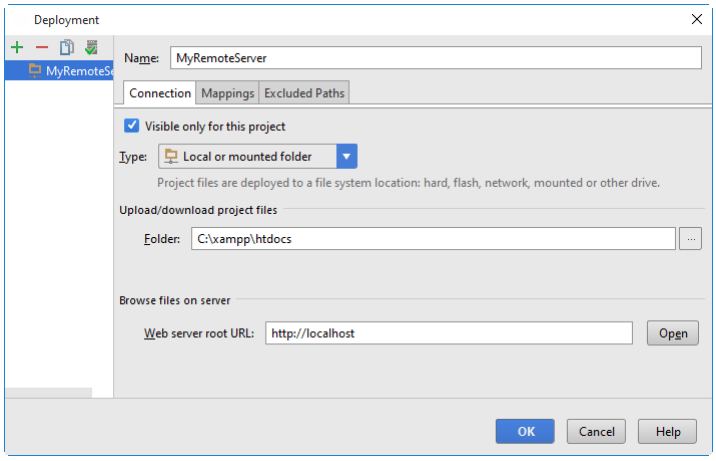
Click +, then in the Add Server dialog box, type your server name (`MyRemoteServer`) and select its type (in our case, this is Local or mounted folder):



OK, the new server is added, but it is still void... It only shows the Web server root URL `http://localhost` , where you will actually browse your uploaded files.

What is specified in the Connection tab?

Select the directory where the project files will be uploaded. In our case, this is the local folder `C:\xampp\htdocs` (You can either type this path manually, or press `Shift+Enter` to open the Select Path dialog .)



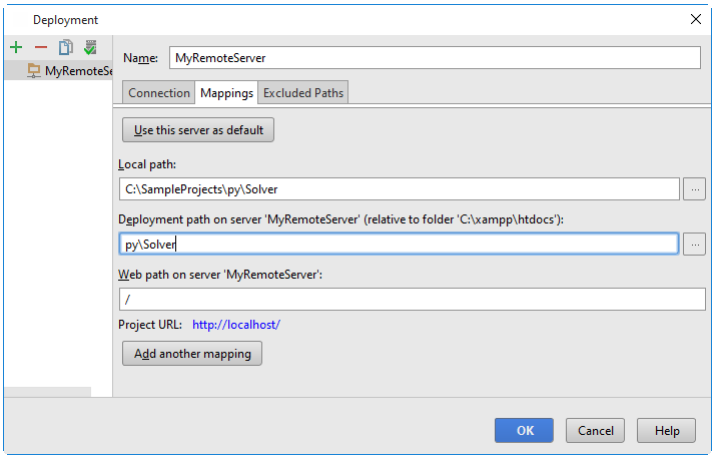
What is specified in the Mapping tab?

Next, choose the Mappings tab . By default, the Local path field contains the project root. However, you can select any other directory within your project tree. Let's assume the default value.

In the Deployment path field (which is by default empty), you have to specify the folder on your server, where IntelliJ IDEA will upload data from the folder, specified in the

Local path(in this example, it's ij\MetersToInchesConverter). This path is specified relative to the folder `C:\xampp\htdocs` !

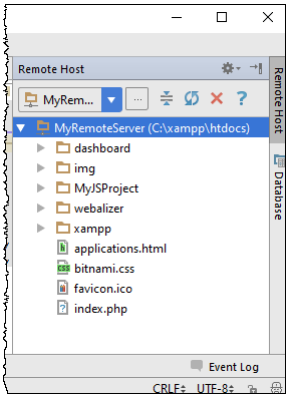
And, finally, let's accept the default value for Web path on server 'MyRemoteServer':



OK, apply changes, and the server is ready to use.

Browsing remote hosts

You can easily make sure your server is up and running. Just choose the command Tools | Deployment | Browse Remote Hosts on the main menu, and the Remote Hosts tool window appears at the right edge of the IntelliJ IDEA's frame:



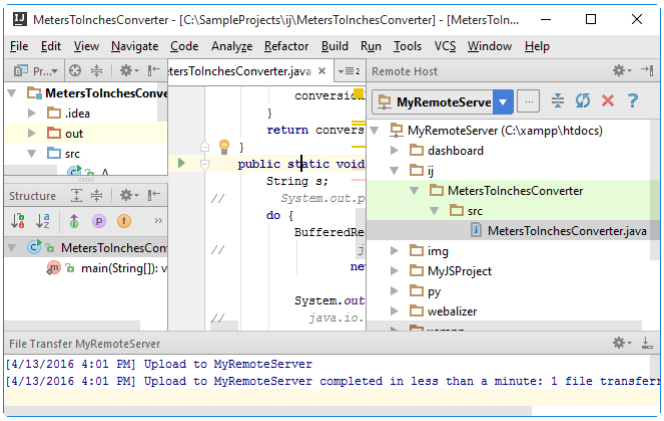
Deployment tools

Next, let's perform some actions, and see what happens.

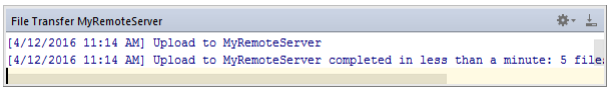
Uploading

First, let's upload one of the files to the remote server. This how it's done...

In the Project tool window , right-click a file you want to upload. In our case, let it be the file `MetersToInchesConverter` . On the context menu, choose Upload to MyRemoteServer, and see the upload results!



You can also upload contents of each directory within your project. For example, right-click the parent directory of the `MetersToInchesConverter` , namely, `src` , choose Upload to MyRemoteServer on the context menu. Wow! We have the entire directory uploaded to the server:

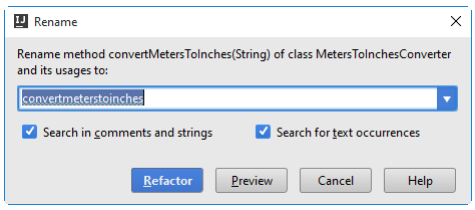


Comparing remote and local versions

There is a local and a remote copy of the file `MetersToInchesConverter` , and they are identical. Let's change the local version.

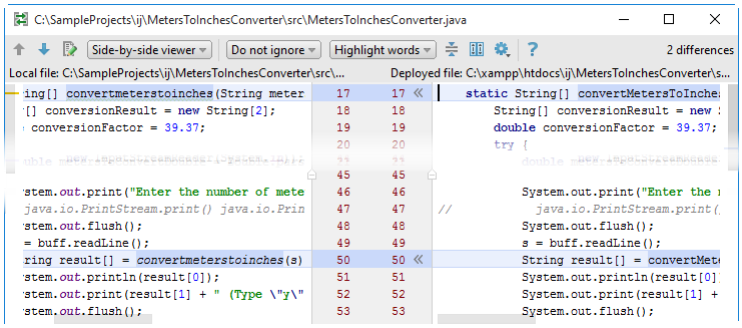
To do that, place the caret at the method declaration, and press `Ctrl+Shift+Alt+T` (or choose Refactor | Refactor This on the main menu). The popup menu shows

all refactorings, available in the current context. Let's choose Rename refactoring, and rename a method :



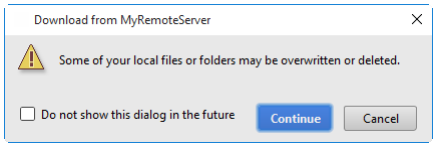
Perform refactoring and see the method name and its usage changed.

OK, now we've changed the local version. Let's make sure IntelliJ IDEA knows about these changes. To do that, again go to the Remote Host Tool Window tool window, and right-click `MetersToInchesConverter`. On the context menu, choose Compare with Local Version. IntelliJ IDEA opens the differences viewer, where you can accept changes or reject them, using the buttons.

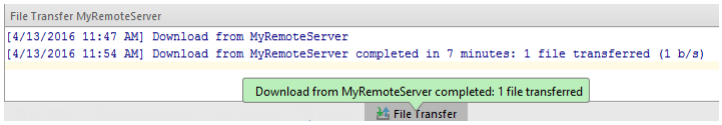


Downloading

In the Remote Host Tool Window tool window, right-click the file `MetersToInchesConverter`, and choose Tools | Deployment | Download from here on the main menu. IntelliJ IDEA immediately shows a warning:



Do not be afraid, and click Continue:

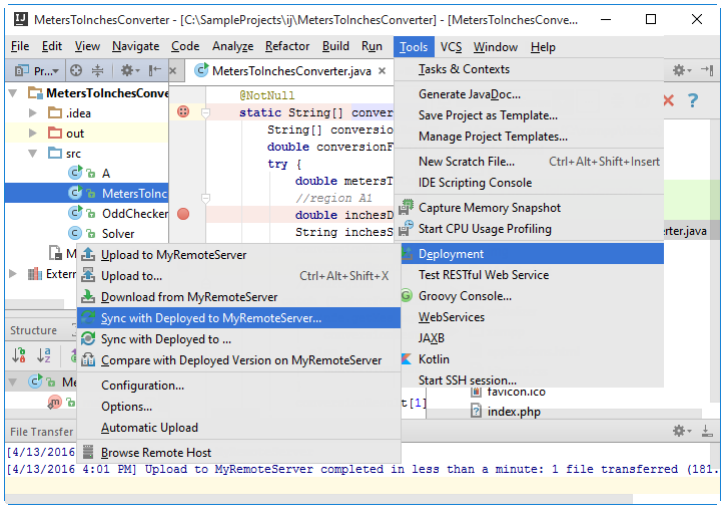


You can also download an entire directory, if it has been previously uploaded to the server. For example, if you click the parent directory `src` and choose the same command, all nested files will be downloaded from the server.

Synchronizing changes

Make a preliminary step - roll the changes to the `MetersToInchesConverter` file back (`Ctrl+Z`). You again see the class `MetersToInchesConverter` with the renamed method.

Next, click `MetersToInchesConverter`, and on the main menu choose Deployment | Sync with Deployed to MyRemoteServer:



IntelliJ IDEA shows differences viewer, where you can accept individual changes or reject them, using the buttons.

Automatic upload to the default server

When a user needs to have the exact same files on the server as in a IntelliJ IDEA project, automatic upload can be of help. Automatic upload means that whenever a change is saved in the IDE, it will be deployed to the default deployment server.

Defining a server as default

A deployment server is considered default, if its settings apply by default during automatic upload of changed files. To define a deployment server as the default one, follow these steps:

1. Choose the desired server in the Deployment page (in our case, `MyRemoteServer`). You can open this page it two possible ways: either Settings/Preferences | Build, Execution, Deployment | Deployment, or Tools | Deployment | Configuration on the main menu.
2. Click .

Enabling automatic upload

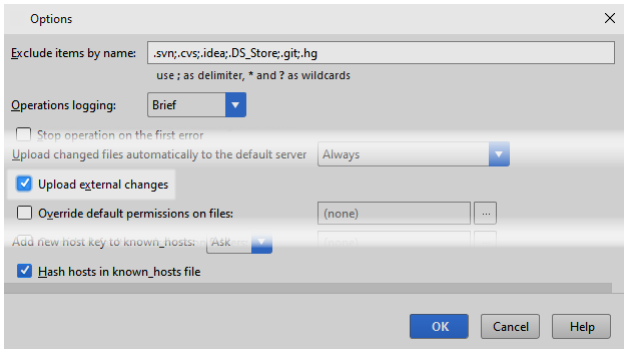
As soon as the default server is set, you can make upload to this server automatic. This can be done in the following two ways:

- First, open the deployment Options (Settings/Preferences | Deployment | Options or Tools | Deployment | Options on the main menu), and in the field Upload files automatically to the default server choose Always, or On explicit save action. The difference between these two choices is explained in the field description .
- Second, on the main menu, select the check command Tools | Deployment | Automatic upload. Note that automatic upload in this case is performed in the Always mode.

It is worth mentioning that the option Always is not recommended for deployment to production: incomplete code can be uploaded while developing, potentially breaking the production application.

Uploading external changes

By default, IntelliJ IDEA uploads only the files changed by itself. If the files are changed by some other process, such as a VCS branch change , transpilation of SASS or LESS or a File Watcher , they are not automatically uploaded. To change this behavior and autoupload these changes as well, enable the Upload external changes option:



Summary

Congrats! You've passed this very basic tutorial. What you've done?

- Created and configured a server of your own.
- Uploaded and downloaded files and folders.
- Compared local and remote versions.
- Configured the server as default.
- Enabled automatic upload of external changes.

Code Coverage

In this section:

- Code Coverage
  - Basics
  - Prerequisite
  - Running with code coverage
- Configuring Code Coverage Measurement
- Running with Coverage
- Viewing Code Coverage Results
- Managing Code Coverage Suites
- Generating Code Coverage Report

Basics

Measuring code coverage is available for testing, applications, and application server run/debug configurations.

The code coverage measuring can be performed using the following runners:

- IntelliJ IDEA code coverage runner (recommended).
- EMMA open-source toolkit. Note that EMMA is not supported by the author any more, and works with Java 7 only when frame validation turned off (pass `-noverify` to the process).
- JaCoCo .

IntelliJ IDEA code coverage runner enables multi-mode analysis:

- Sampling mode enables collecting line coverage with negligible slow-down.
- Tracing mode enables accurate collection of the branch coverage, with the ability to track tests, view coverage statistic , and get additional information on each covered line.

For the other runners, only sampling mode is available.

Code coverage results are reflected in the dedicated Coverage tool window , in the Project view of the Project tool window , and in the editor. The tool windows show the following information:

- For a directory: the percentage of the covered classes and lines.
- For a class: the percentage of the covered methods and lines.

When a file is opened in the editor, each line is highlighted with regard to its code coverage status:

- Lines executed during simulation are marked green.
- Lines not executed during simulation are marked red.
- Covered lines with conditions are marked yellow in the tracing mode.

The coverage measurement results comprise a coverage suite. You can have the results of a new simulation merged with any existing suite. In this case, a line will be considered covered if it is covered by at least one of the simulations.

A coverage suite is generated every time a test or application with code coverage measurement is executed. It is possible to have an unlimited amount of coverage suites.

Prerequisite

Make sure the Code Coverage plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins . If the plugin is disabled, the code coverage tabs will not be visible in the run/debug configuration dialogs.

To use code coverage in project, follow these general steps

1. Specify how you want to process the coverage results .
2. Create tests for the target code, if you are going to measure code coverage for testing.
3. Configure code coverage measurement in the desired run/debug configuration.
4. Run with coverage , using the dedicated command on the main menu Run | Run with Coverage, or .
5. Once the run with coverage has been executed, you can
   - Use the various coverage suites .
   - View code coverage data .
   - Generate code coverage report .

Configuring Code Coverage Measurement

IntelliJ IDEA makes it possible to configure the various aspects of code coverage measurement. In this section:

- Configuring the way coverage suites are processed
- Configuring code coverage measurement options
- Changing colors of the coverage highlighting

To configure code coverage behavior

1. Open the Settings dialog box , and then click Coverage under Build, Execution, Deployment. The Coverage page opens.

2. Define how the collected coverage data will be processed:
   - To have the Code Coverage dialog box shown every time you launch a new run configuration with code coverage, choose Show options before applying coverage to the editor.
   - To discard the new code coverage results, choose Do not apply collected coverage.
   - To discard the active suites and use the new one every time you launch a new run configuration with code coverage, choose Replace active suites with the new one.
   - To have the new code coverage suite appended to the active suites every time you launch a new run configuration with code coverage choose Add to active suites.

3. Define the behaviour of the Coverage tool window when an application or test is run with coverage:
   - To have the Coverage tool window opened automatically, select the Activate Coverage View check box.
   - To open the Coverage tool window manually, clear the Activate Coverage View check box.

To configure code coverage options

1. Open the Edit Run/Debug Configuration dialog box, add the desired run/debug configuration, and click the Code Coverage tab.

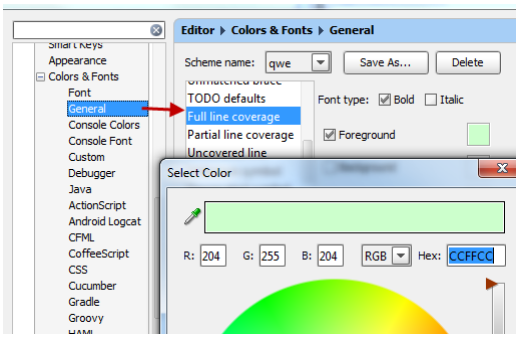
2. In the Code Coverage tab, define the following options:
   - From the Choose coverage runner drop-down list, select the desired code coverage runner. The available options are:
     - EMMA
     - IntelliJ IDEA
   - Choose the options for the selected runner:
     - For the EMMA runner, only the Sampling mode is available.
     - For the IntelliJ IDEA runner, you can choose between the Sampling or Tracing modes.
   - Specify the scope to measure code coverage for. Do one of the following:
     - To specify a class, click the Add Class button.
     - To specify a package, click the Add Package button.
   - To have code coverage statistic collected for folders with tests as well, select the Enable Coverage in Test Folders check box.

To configure code coverage colors

1. Open the Editor - Colors and Fonts page of the Settings dialog.
   Alternatively, just click in the statistics pop-up.

2. Expand Colors and Fonts node, and select General.

3. In the list of textual components, select the required type of coverage, for example, Full line coverage or Uncovered line, and then choose the desired colors:



Running with Coverage

IntelliJ IDEA provides a dedicated action that allows you to perform run with code coverage measurement. The code coverage data are processed according to the option selected in the Coverage page of the Settings dialog box.

To run with code coverage measurement

1. Do one of the following:
   - Open the desired file in the editor, and choose Run <name> with coverage on the context menu. When running tests with coverage, note that you can run the entire test class, or each individual test method, depending on the caret location.
   - Select the desired run/debug configuration, and then on the main menu choose Run | Run <run/debug configuration name> with coverage.
   - On the main toolbar, click . This will launch the selected run/debug configuration.

2. If the Show options before applying coverage to the editor check box has been selected in the Coverage page of the Settings dialog box, choose whether you want to replace the active coverage suites, or add the collected data to the active suites, or do not want not apply coverage data. You can also opt to skip this dialog in the future.
   In case any other option has been selected, the respective action will be performed silently.

3. Explore the collected coverage data in the Coverage tool window .

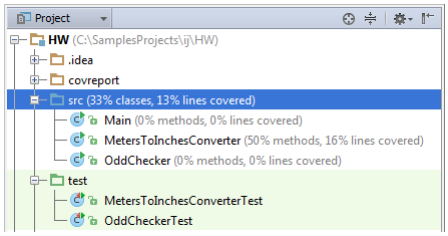
Viewing Code Coverage Results

Viewing code coverage helps you detect pieces of your source code that are not affected by simulation.

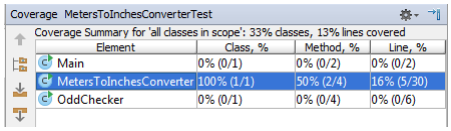
To view code coverage results

1. Do one of the following:
   - Run the desired class with coverage , select suite to show , and open class in the editor.
   - On the main menu, choose Analyze | Show Code Coverage Data .
   - Press `Ctrl+Alt+F6` .

2. View coverage results:
   - In the Project tool window:

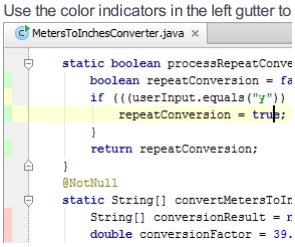


   - In the dedicated Coverage tool window :

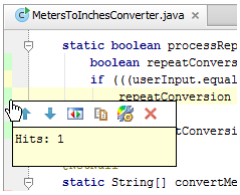


3. Open in the editor the files you want to explore.

4. Use the color indicators in the left gutter to detect the uncovered lines of code.



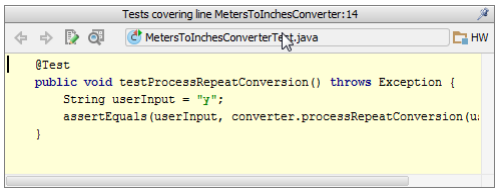
5. To find out how many times a line has been hit, click the line in the gutter area.



The pop-up window that opens shows the statistic for the line at caret. For lines with conditions, the pop-up window also provides statistic on the number of true and false hits.

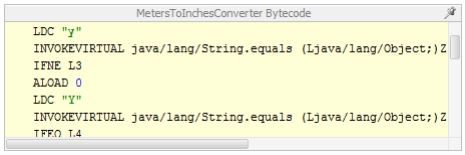
Use the following toolbar buttons:

- : jump to the next/previous groups of covered or uncovered lines.
- : view JUnit tests that cover the line at caret.
  The test that covers the line at caret, is shown in a pop-up window:



  - This button is only available in the Tracing mode, and with the Track per test coverage check box selected.
  - When pinned, this pop-up window converts into the Find tool window .

- : show byte code of the current class in a pop-up window:



  - This button is only available, when Byte Code Viewer plugin that comes bundled with the product, is enabled.
  - When pinned, this pop-up window converts into the Byte Code Viewer.

- : open the General page of the Colors and Fonts settings, with the line Full line coverage highlighted.

- ✖: hide coverage information.

6. Create missing tests, as described in the section Creating Tests .

Managing Code Coverage Suites

IntelliJ IDEA provides a tool to select coverage suites for showing or hiding, adding and removing suites.

In this section:

– Accessing the Choose Coverage Suite to Display dialog box
– Selecting coverage suites to show
– Hiding coverage suites
– Adding coverage suites from disk

To open the Choose Coverage Suite to Display dialog box, do one of the following

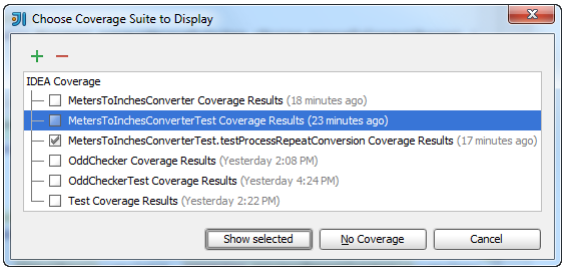
– On the main menu, choose Analyze | Show Coverage Data.

– Press `Ctrl+Alt+F6`.

To select coverage suites to show

1. Open the Choose Coverage Suite to Display dialog box, as described above.

2. In the Choose Coverage Suite to Display dialog box, select the check boxes next to the desired suites.



3. Click Show selected. The dialog box closes.

4. Open in the editor the classes, for which the coverage suites have been selected, and explore the coverage results.

To hide coverage suites

1. Open the Choose Coverage Suite to Display dialog box, as described above.

2. In the Choose Coverage Suite to Display dialog box, select the check boxes next to the suites you want to hide.

3. Click No coverage. The dialog box closes. The coverage results are not shown for the target classes.

To add or delete a coverage suite

Consider a situation when a file that contains code coverage information, has been obtained from the build server. You can load this file from disk and make it available for review. Also, you can bring for examination the coverage suite that has been produced some time ago.

On the other hand, IntelliJ IDEA allows you to remove unnecessary coverage suites.

1. Open the Choose Coverage Suite to Display dialog box, as described above.

2. Add new suites or delete the existing ones:
   – Click +, and select the desired `*.es` file in the file chooser dialog.
   – Select one or more suites in the list, and click –. The selected suite will be deleted from the list, and from storage.

Generating Code Coverage Report

IntelliJ IDEA suggests two ways of generating HTML reports on the base of the code coverage measurement results: using the menu command, or using the Coverage tool window .

To generate a code coverage report

1. Do one of the following:
   - On the main menu, choose Analyze | Generate Coverage Report.
   - In the toolbar of the Coverage tool window , click .
2. In the Generate Coverage Report dialog box that opens, specify the target directory where the generated report will be stored, and optionally select the check box Open generated HTML in browser.
3. Click Save. IntelliJ IDEA will store the generated report to the specified location, and also open it in the default browser, if the corresponding check box has been selected.

Please note that if you are going to save a code coverage report for one of the multiple projects opened in the same window , it is important to check the suggested target location, because IntelliJ IDEA suggests the previously used location.

Refactoring Source Code

IntelliJ IDEA offers a wide variety of code refactorings, which track down and correct the affected code references automatically.

In this part you will find:

- General refactoring procedure
- Procedures and examples of the available refactorings

To perform refactoring, follow these general steps

1. Select (or hover caret on) a symbol or code fragment to refactor. The set of available refactorings depends on your selection. You can select symbols in the following IntelliJ IDEA components:
   - Project view
   - Structure tool window
   - Editor
   - UML Class diagram

2. Do one of the following:
   - On the main Refactor menu or on the context menu of the selection, choose the desired refactoring or press the corresponding keyboard shortcut (if any).
   - On the main menu, choose Refactor | Refactor This, or press `Ctrl+Shift+Alt+T`, and then select the desired refactoring from the pop-up window.

3. In the dialog box that opens, specify the refactoring options.

4. To apply the changes immediately, depending on the refactoring type, click Refactor or OK.

5. For certain refactorings, there is an option of previewing the changes prior to actually performing the refactoring. In such cases the Preview button is available in the corresponding dialog.
   To preview the potential changes and make the necessary adjustments, click Preview. IntelliJ IDEA displays the changes that are going to be made on a dedicated tab of the Find tool window .

   One of the possible actions at this step is to exclude certain entries from the refactoring. To do so, select the desired entry in the list and press `Delete`.

   If conflicts are expected after the refactoring, IntelliJ IDEA displays a dialog with a brief description of the encountered problems. If this is the case, do one of the following:
   - Ignore the conflicts by clicking the Continue button. As a result, the refactoring will be performed, however, this may lead to erroneous results.
   - Preview the conflicts by clicking the Show in View button. IntelliJ IDEA shows all conflicting entries on the Conflicts tab in the Find tool window , enabling you to navigate to the problematic lines of code and to make the necessary fixes.
   - Cancel the refactoring and return to the editor.

6. When you are satisfied with the proposed results, click Do Refactor to apply the changes.

IntelliJ IDEA provides the following common refactorings:

- Change Class Signature
- Change Signature
- Convert Anonymous to Inner
- Convert to Instance Method
- Copy
- Encapsulate Fields
- Extract Refactorings
- Generify Refactoring
- Inline
- Invert Boolean
- Make Class Static
- Make Method Static
- Migrate
- Move Refactorings
- Pull Members up
- Push Members down
- Remove Middleman
- Rename Refactorings
- Replace Constructor with Builder
- Replace Constructor with Factory Method
- Replace Inheritance with Delegation
- Find and Replace Code Duplicates
- Replace Temp with Query
- Safe Delete
- Type Migration
- Use Interface Where Possible
- Wrap Return Value
- Tutorial. Replace Conditional Logic with Strategy Pattern

The following language-specific refactorings are available:

- Change Signature in JavaScript

- Change Signature in JavaScript
- Extract Parameter in JavaScript
- Extract Variable in JavaScript

Change Class Signature

The Change Class Signature refactoring lets you turn a class into a generic and manipulate its type parameters. The refactoring automatically corrects all calls, implementations and overridings of the class.

- Example
- Use cases
- Changing a class signature

Example

Before **After**

```
// This is the class whose signature will be changed:

                    public class MyClass {
                    // some code here
                    }

                    public class MyOtherClass {

                    // Here are the references to MyClass:

                    MyClass myClass;
                    void myMethod(MyClass myClass) {
                    // some code here
                    }
                    // some code here
                    }

                    // Now we are going to add two formal type parameters
                    // to MyClass.
```

```
// Two formal type parameters have been added:

                    public class MyClass<Param1, Param2> {
                    // some code here
                    }

                    public class MyOtherClass {

                    // The references to MyClass have changed accordingly:

                    MyClass<String, Integer> myClass;
                    void myMethod(MyClass<String, Integer> myClass) {
                    // some code here
                    }
                    // some code here
                    }

                    // When performing the refactoring, String and Integer
                    // were specified as the default values for Param1 and
                    // Param2 respectively.
```

Use cases

- The type parameters can have bounds extending other constructs including other type parameters from another class. For instance, the code like `public class ClassA<T, E>` can be easily transformed to `public class ClassA<T, E extends T>` . However, if the parameter order was initially `public class ClassA<E, T>` , the Change Class Signature refactoring may be useful for changing the parameter order in the class signature as well as in all its usages.
- IntelliJ IDEA can successfully deal with rather complicated cases and can alter parameters and corresponding usages if one parameter depends on another.

Changing a class signature

To change a class signature, follow these steps

1. In the editor, place the cursor within the name of the class whose signature you want to change.
2. Do one of the following:
   - Press `Ctrl+F6` .
   - Choose Refactor | Change Signature in the main menu.
   - Select Refactor | Change Signature from the context menu.
3. In the Change Class Signature dialog , use the available controls to manage the formal type parameters:
   - To add a new parameter, click + or press `Alt+Insert` . Specify the parameter name and default type in the Name and the Default Value fields respectively.
   - To remove a parameter, select the parameter and click – (`Alt+Delete`).
   - To move a parameter up or down in the list, select the parameter and then use ↑ (`Alt+Up`) or ↓ (`Alt+Down`).
4. To perform the refactoring right away, click Refactor.
   To see the expected changes and make the necessary adjustments prior to actually performing the refactoring, click Preview.

Change Signature

In this section:

- Basics
- Example
- Initializer, default value, and propagation of new parameters
- More refactoring examples
- Changing a method signature

Basics

The Change Signature refactoring combines several different modifications that can be applied to a method signature. You can use this refactoring for the following purposes:

- To change the method name.
- To change the method return type.
- To add new parameters and remove the existing ones.
- To assign default values to the parameters.
- To reorder parameters.
- To change parameter names and types.
- To propagate new parameters through the method call hierarchy.

When changing a method signature, IntelliJ IDEA searches for all usages of the method and updates all the calls, implementations, and override replacements of the method that can be safely modified to reflect the change.

Note The Change Method Signature refactoring is supported for Java, PHP, JavaScript and ActionScript.
- You can access this refactoring from a UML Class diagram .
- The Change Method Signature refactoring is applicable to a constructor. In this case, however, the name and the return type cannot be changed.

Example

Before After

```
// The function paint() is declared in
                    // the IShape interface.



                    public interface IShape {
                    function paint(g:
                    Graphics): void;
                    }

                    // This function is then called within the
                    // paint() function of the Canvas class.



                    public class Canvas {
                    private var shapes: Vector.<IShape>;

                    public function paint(g: Graphics): void
                    {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g);
                    }
                    }
                    }

                    // Now we are going to show an example of the
                    // Change Signature refactoring for the function
                    // paint() of the IShape interface.
```

```
// In this refactoring example we have changed the name of the existing
                    parameter
                    // and introduced two new parameters. Note that the first of the new parameters
                    // a required parameter while the second is optional because the default value
                    // for it is specified in the function definition.

                    public interface IShape {
                    function paint(graphics:Graphics, wireframe:Boolean, offset:Point
                    = null):void;
                    }

                    // When performing this refactoring, the new parameters were propagated to
                    // the paint() function of the Canvas class. As a result, the signature of
                    // Canvas.paint() has changed. Also note how IShape.paint() within
                    // Canvas.paint() is called now.

                    public class Canvas {
                    private var shapes: Vector.<IShape>;

                    public function paint(g:Graphics, wireframe:Boolean): void {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g, wireframe);
                    }
                    }
                    }
                    // Other results for this refactoring are possible.
                    // For more information, see the discussion that follows.
```

Initializer, default value, and propagation of new parameters

For each new parameter added to a method, you can specify:

- A value (or an expression) to be used for initializing the parameter (the Initializer field in IntelliJ IDEA).
- A default value (or an expression) (the Default value field).

You can also propagate the parameters you have introduced to the methods that call the function whose signature you are changing.

The refactoring result depends on whether you specify the default value and whether you use propagation.

Propagation. New parameters can be propagated to any method that calls the method whose signature you are changing. In such case, generally, the signatures of the calling methods change accordingly.

These changes, however, also depend on the combination of the initializer and the default value set for the new parameters .

Initializer. The value specified in the Initializer field is added to the function definition as the default parameter value. This makes the corresponding parameter an optional parameter. (See the discussion of required and optional parameters in Flex/ActionScript documentation .)

If the default value for the new parameter is not specified (in the Default value field), irrespective of whether or not the propagation is used, the method calls and the signatures of the calling methods don't change.

If both, the initializer and the default value are specified, the refactoring result depends on whether or not the propagation is used:

- If the propagation is not used, the initializer value don't affect the function calls and the signatures of the calling functions.
- If the propagation is used, the initializer value is added to the definition of the calling function as the default value for the corresponding parameter (in the same way as in the function whose signature you are changing).

Default value. Generally, this is the value to be added to the method calls.

If the new parameter is not propagated to a calling method, the calls within such method will also use this value.

If the propagation is used, this value won't matter for the method calls within the calling methods.

More refactoring examples

To see how different refactoring settings discussed above affect the refactoring result, let us consider the following examples.

All the examples are a simplified version of the refactoring shown earlier . In all cases, a new parameter `wireframe` of the type `Boolean` is added to the function `paint()` defined in the `IShape` interface.

In different examples, different combinations of the initializer and the default value are used, and the new parameter is either propagated to `Canvas.paint()` (which calls `IShape.paint()`) or not.

Initializer	Default value	Propagation used	Result
	false	Yes	

```
public interface IShape
                    {
                    function paint(g:Graphics, wireframe:Boolean):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics, wireframe:Boolean): void
                    {
                    for each
                    (
                    var shape: IShape in
                    shapes) {
                    shape.paint(g,wireframe);
                    }
                    }
```

Initializer	Default value	Propagation used	Result
	false	No	

```
public interface IShape
                    {
                    function paint(g:Graphics, wireframe:Boolean):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics): void
                    {
                    for each
                    (
                    var shape: IShape in
                    shapes) {
                    shape.paint(g,false);
                    }
                    }
```

true		Yes	

```
public interface IShape
                    {
                    function paint(g:Graphics, wireframe:Boolean
                    = true):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics):
                    void
                    {
                    for each
                    (
                    var shape: IShape in
                    shapes) {
                    shape.paint(g);
                    }
                    }
```

true		No	

```
public interface IShape
                    {
                    function paint(g:Graphics, wireframe:Boolean
                    = true):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics):void
                    {
                    for each
                    (
                    var shape: IShape in
                    shapes) {
                    shape.paint(g);
                    }
                    }
```

true	false	Yes	

```
public interface IShape
                    {
                    function paint(g:Graphics, wireframe:Boolean
                    = true):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics, wireframe:Boolean = true):
                    void {
                    for each
                    (
                    var shape: IShape in
                    shapes) {
                    shape.paint(g,wireframe);
                    }
                    }
```

true	false	No	

```
public interface IShape
                    {
                    function paint(g:Graphics,
                    wireframe:Boolean = true):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics):
                    void {
                    for each
                    (
                    var shape: IShape in
                    shapes) {
                    shape.paint(g,false);
                    }
                    }
```

The following table shows 3 different ways of performing the same Change Signature refactoring.

In all the cases, the function `result()` is renamed to `generate_result()` and a new parameter `input` is added to this function.

The examples show how the function call, the calling function (`show_result()`) and other code fragments may be affected depending on the refactoring settings.

Before After

```
// This is the function whose signature will be changed:
                    function doSomething($a) {
                    // some code here
                    }

                    function refactor($a)
                    {

                    // Here is the function call:
                    $this->doSomething($a);
                    }

                    // Now we'll rename the function and
                    // add one parameter.
```

```
// The function has been renamed to doSomeRefactoring.
                    // The new parameter $b has been added.

                    function doSomeRefactoring($a,$b) {
                    // some code here
                    }

                    function refactor($a,$b) {

                    // The function call has been changed accordingly:
                    $this->doSomeRefactoring($a,$b);
                    }
```

```
// This is the function whose signature will be changed:


                    function doSomething($a) {
                    // some code here
                    }

                    function refactor($a)
                    {

                    // Here is the function call:
                    $this->doSomething($a);
                    }

                    // Now we'll rename the function and add one parameter.
                    //We will also specify the default value 'new_param'
                    //for this new parameter
```

```
// The function has been renamed to doSomeRefactoring.
                    // The new parameter $b has been added.

                    function doSomeRefactoring($a,$b) {
                    // some code here
                    }

                    function refactor($a)
                    {

                    // The function call has been changed accordingly:
                    $this->doSomeRefactoring($a,'new_param');
                    }

                    // When performing the refactoring, 'new_param' was specified as
                    // the default parameter value.
```

```
// This is the function whose signature will be changed:


                    function doSomething($a) {
                    // some code here
                    }


                    function refactor($a)
                    {

                    // Here is the function call:
                    $this->doSomething($a);
                    }

                    // Now we'll rename the function and add one parameter.
                    //We will also propagate this new parameter
                    //through the calling function refactor() to the function call.
```

```
// The function has been renamed to doSomeRefactoring.
                    // The new parameter $b has been added.

                    function doSomeRefactoring($a,$b) {
                    // some code here
                    }

                    // Note the new function parameter:
                    function refactor($a,$b) {

                    // The function call has been changed accordingly:
                    $this->doSomeRefactoring($a,$b);
                    }
```

To change a method signature

1. In the editor, place the cursor within the name of the method whose signature you want to change.

2. Do one of the following:
   - Press `Ctrl+F6`.
   - Choose Refactor | Change Signature on the main menu or .
   - Choose Refactor | Change Signature on the context menu.

3. In the Change Signature dialog, make the necessary changes to the method signature and specify what other, related, changes are required.
   You can:
   - Change the method name. To do that, edit the text in the Name field.
   - Change the method return type by editing the contents of the Return type field.
   - Manage the method parameters using the table and the buttons in the Parameters area:
     - To add a new parameter, click + and specify the properties of the new parameter in the corresponding table row.
       When adding parameters, you may want to propagate these parameters to the methods that call the current method.
     - To remove a parameter, click any of the cells in the corresponding row and click −.
     - To reorder the parameters, use the ↑ and ↓ buttons. For example, if you wanted to put a certain parameter first in the list, you would click any of the cells in the row corresponding to that parameter, and then click ↑ the required number of times.
     - To change the name, type, the initializer, or the default value of a parameter, make the necessary edits in the table of parameters (in the fields Name Type, Initializer and Default value respectively).
   - Propagate new method parameters (if any) along the hierarchy of the methods that call the current method.

(There may be methods that call the method whose signature you are changing. These methods, in their turn, may be called by other methods, and so on. You can propagate the changes you are making to the parameters of the current method through the hierarchy of calling methods and also specify which calling methods should be affected and which shouldn't.)

To propagate a new parameter:

1. Click the Propagate Parameters button .
2. In the left-hand pane of the Select Methods to Propagate New Parameters dialog, expand the necessary nodes and select the check boxes next to the methods you want the new parameters to be propagated to.
   To help you select the necessary methods, the code for the calling method and the method being called is shown in the right-hand part of the dialog (in the Caller Method and Callee Method panes respectively).

   As you switch between the methods in the left-hand pane, the code in the right-hand pane changes accordingly.
3. Click OK.

4. To perform the refactoring right away, click Refactor.
   To see the expected changes and make the necessary adjustments prior to actually performing the refactoring, click Preview.

Note Code completion is available in the Default value field of the table in the Parameters area.

Change Method Signature in Java

In Java, you can use the Change Method Signature refactoring to:

- Change the method name, return type and visibility scope.
- Add new parameters and remove the existing ones. Note that you can also add a parameter using a dedicated Extract Parameter refactoring.
- Reorder parameters.
- Change parameter names and types.
- Add and remove exceptions.
- Propagate new parameters and exceptions through the method call hierarchy.

On this page:

- Examples
- Changing a method signature

Examples

The following table shows 4 different ways of performing the same Change Method Signature refactoring.

In all the cases, a new parameter `price` of the type `double` is added to the method `myMethod()`.

The examples show how the method call, the calling method (`myMethodCall()`) and other code fragments may be affected depending on the refactoring settings.

Before After

```
public class MyClass
                    {

                    // This is the method whose signature will be changed:

                    public void myMethod(int value) {
                    // some code here
                    }
                    }

                    public class MyOtherClass {
                    public void myMethodCall(MyClass myClass) {
                    double d=0.5;

                    // Here is the method call:

                    myClass.myMethod(1);
                    }
                    }

                    // We'll ask IntelliJ IDEA to update all the method calls.
                    // We'll also specify a default value to be passed to the method.
```

```
public class MyClass
                    {

                    // The new parameter price has been added:

                    public void myMethod(int i, double price) {
                    // some code here
                    }
                    }

                    public class MyOtherClass {
                    public void myMethodCall(MyClass myClass) {
                    double d=0.5;

                    // The method call has changed accordingly:

                    myClass.myMethod(1, 0.0);
                    }
                    }

                    // When performing the refactoring, 0.0 was specified as
                    // the default parameter value.
```

```
public class MyClass
                    {

                    // This is the method whose signature will be changed:

                    public void myMethod(int value) {
                    // some code here
                    }
                    }

                    public class MyOtherClass {
                    public void myMethodCall(MyClass myClass) {
                    double d=0.5;

                    // Here is the method call:

                    myClass.myMethod(1);
                    }
                    }

                    // We'll ask IntelliJ IDEA to update all the method calls.
                    // We'll also ask IntelliJ IDEA to look for a variable
                    // of the appropriate type near the method call and pass this
                    // variable to the method.
                    // In IntelliJ IDEA, this option is called Use Any Var.
```

```
public class MyClass
                    {

                    // The new parameter price has been added:

                    public void myMethod(int i, double price) {
                    // some code here
                    }
                    }

                    public class MyOtherClass {
                    public void myMethodCall(MyClass myClass) {
                    double d=0.5;

                    // The method call has changed accordingly:

                    myClass.myMethod(1, d);
                    }
                    }

                    // Near the method call, IntelliJ IDEA has found the variable d
                    // which has the same type as the new parameter. As a result,
                    // this variable was used in the method call.
```

```java
public class MyClass
                    {









                    // This is the method whose signature will be changed:

                    public void myMethod(int value) {
                    // some code here
                    }
                    }

                    public class MyOtherClass {
                    public void myMethodCall(MyClass myClass) {
                    double d=0.5;

                    // Here is the method call:

                    myClass.myMethod(1);
                    }
                    }

                    // We'll ask IntelliJ IDEA to keep the method calls unchanged but
                    // create a new overloading method which will call the method
                    // with the new signature.
                    // In IntelliJ IDEA, this way of handling the method calls is
                    // referred to as Delegate via overloading method.
```

```java
public class MyClass
                    {

                    // A new overloading method has been created:

                    public void myMethod(int i) {
                    myMethod(i, 0.0);
                    }

                    // The new parameter price has been added:

                    public void myMethod(int i, double price) {
                    // some code here
                    }
                    }

                    public class MyOtherClass {
                    public void myMethodCall(MyClass myClass) {
                    double d=0.5;

                    // The method call has not changed:

                    myClass.myMethod(1);
                    }
                    }

                    // Note that the new overloading method has the old signature.
                    // However, it calls the method with the new signature.
                    // 0.0 was specified as the default parameter value
                    // when performing the refactoring.
```

```
public class MyClass
                    {

                    // This is the method whose signature will be changed:

                    public void myMethod(int value) {
                    // some code here
                    }
                    }

                    public class MyOtherClass {

                    // This method will also change its signature:


                    public void myMethodCall(MyClass myClass) {
                    double d=0.5;

                    // Here is the method call:

                    myClass.myMethod(1);
                    }
                    }

                    // This time we'll ask IntelliJ IDEA to propagate the new
                    // parameter to the method call through the calling method
                    // myMethodCall().
```

```
public class MyClass
                    {

                    // The new parameter price has been added:

                    public void myMethod(int i, double price) {
                    // some code here
                    }
                    }

                    public class MyOtherClass {

                    // The new parameter price has propagated
                    // to the method call through the calling method:

                    public void myMethodCall(MyClass myClass, double price) {
                    double d=0.5;

                    // The method call has changed accordingly:

                    myClass.myMethod(1, price);
                    }
                    }
```

Changing a method signature

1. In the editor, place the cursor within the name of the method whose signature you want to change.

2. Do one of the following:
   - Press `Ctrl+F6`.
   - Choose Refactor | Change Signature in the main menu.
   - Select Refactor | Change Signature from the context menu.

   Note that if you refactor a method that overrides another method, IntelliJ IDEA suggests either to modify the method from the base class, or to modify only the selected method.

3. In the Change Signature dialog , make the necessary changes to the method signature and specify which other, related changes are required.
   You can:
   - Change the method visibility scope (access level modifier) by selecting the necessary option under Visibility.
   - Change the method return type by editing the contents of the Return type field.
     Note that code completion is available in this field, and also in other fields used for specifying the types.
   - Change the method name. To do that, edit the text in the Name field.
   - Manage the method parameters using the controls on the Parameters tab:
     - To add a new parameter, click + (`Alt+Insert`) and specify the properties of the new parameter in the corresponding fields. If necessary, select the Use Any Var option .
       When adding parameters, you may want to propagate these parameters to the methods that call the current method.
     - To remove a parameter, select this parameter in the table and click – (`Alt+Delete`).
     - To reorder the parameters, use ↑ (`Alt+Up`) and ↓ (`Alt+Down`).
     - To change the name, type, or the default value for a parameter, click this parameter in the table and make the necessary edits in the corresponding fields.
   - Propagate new method parameters (if any) along the hierarchy of the methods that call the current method.
     (There may be the methods that call the method whose signature you are changing. These methods, in their turn, may be called by other methods, and so on. You can propagate the changes you are making to the parameters of the current method through the hierarchy of the calling methods and also specify which calling methods should be affected and which shouldn't.)

     To propagate the new parameters:
     1. Click (`Alt+G`).
     2. In the left-hand pane of the Select Methods to Propagate New Parameters dialog, expand the necessary nodes and select the check boxes next to the methods you want the new parameters to be propagated to.
        To help you select the necessary methods, the code for the calling method and the method being called is shown in the right-hand part of the dialog (in the Caller Method and Callee Method panes respectively).

        As you switch between the methods in the left-hand pane, the code in the right-hand part changes accordingly.
     3. Click OK.
   - Manage the method exceptions using the list of exception types and the buttons on the Exceptions tab. The procedures are similar to those used for managing the method parameters .
   - Propagate new exceptions (if any) along the hierarchy of the methods that call the current method. To initiate this procedure, use (`Alt+X`). In all other respects, the procedure is similar to that used for propagating new method parameters .
   - Specify how the method calls should be handled. To do that, select one of the following Method calls options:

- If you want the method calls to be modified , select Modify.
- If you want to leave the existing method calls unchanged , select Delegate via overloading method.

4. To perform the refactoring right away, click Refactor.
   To see the expected changes and make the necessary adjustments prior to actually performing the refactoring, click Preview.

Convert Anonymous to Inner

The Convert Anonymous to Inner refactoring allows you to convert an anonymous class into a named inner class.

– Example

– To inline a constructor, follow these steps

Example

Before After

```
public class Class
                    {
                    public Interface method() {
                    final int i = 0;
                    return new Interface() {
                    public int publicMethod() {
                    return i;}
                    };
                    }
                    }
```

```
public class Class
                    {
                    public Interface method() {
                    final int i = 0;
                    return new MyInterfaceClass(i);
                    }
                    }
                    public class MyInterfaceClass implements
                    Interface {
                    private final int
                    i;
                    public MyInterfaceClass(int
                    i) {
                    this.i = i;
                    }
                    public int publicMethod() {
                    return
                    i;
                    }
                    }
```

To inline a constructor, follow these steps

1. Place the cursor within the anonymous class to be refactored.
2. On the main menu, or on the context menu of the selection, choose Refactor | Convert Anonymous To Inner . The Convert Anonymous To Inner dialog opens.
3. In the Class name field specify the name for the new inner class.
4. In the Constructor parameters area select the variables, that will be used as parameters to the inner class constructor.
5. Click OK to create the inner class.

Convert to Instance Method

Convert to Instance Method refactoring is used to convert a static method to a non-static, class instance method, with a class being the type parameter of the initial method.

– Example

– Converting a method to an instance method

Tip This refactoring is also available from UML Class diagram .

Example

Consider classes `MyClass` , `ClassB` and `ClassB` residing in the same package. `MyClass` contains the following code:

```
public class MyClass {
    ClassA classA = new ClassA();
    ClassB classB = new ClassB();

    static public void greatMethod(ClassA classA, ClassB classB){
        System.out.println("classA = " + classA);
        System.out.println("classB = " + classB);
    }

    public void myMethod(){
        MyClass.greatMethod(classA, classB);
    }
}
```

After refactoring, `MyClass` and `ClassB` become:

```
public class MyClass {
    ClassA classA = new ClassA();
    ClassB classB = new ClassB();

    public void myMethod(){
        classB.greatMethod(classA);
    }
}

public class ClassB {
    public void greatMethod(ClassA classA){
        System.out.println("classA = " + classA);
        System.out.println("classB = " + this);
    }
}
```

Converting a method to an instance method

1. In the editor, place the caret on the declaration or usage of a method to be refactored. The method should be static and the types of its parameters should be the classes from the project. In other words, you cannot use such parameters types, as `String` .
2. Do one of the following:
   – On the Refactor menu, choose Convert to Instance Method.
   – Right-click the method and select Refactor | Convert to Instance Method .

   Note The selected method must be static and must receive at least one of the classes included in the current project as a parameter.
3. The Convert to Instance Method dialog appears.
4. In the Select an instance parameter list select the class you want the method to belong to after the conversion. All the usages of this class inside the method are replaced with this.
5. To change the visibility scope of the converted method, select the new scope in the Visibility area. By default the converted method will have no scope declaration (equivalent to public).
6. Preview and apply changes .

Copy

– Basics
– Performing Copy refactoring

Basics

The Copy refactoring lets you create a copy of a class in a different package. It can also be used to create a copy of a file, directory or package in a different directory or package.

Performing Copy refactoring

To perform the Copy refactoring

1. Select the item of interest in a tool window (e.g. the Project tool window). Alternatively, open the necessary class or file in the editor.
2. Do one of the following:
   – Choose Refactor | Copy on the main menu or the context menu.
   – Press `F5`.
   – In the Project tool window, press and hold the `Ctrl` key, and drag the item to the target location.
3. In the Copy dialog that opens, specify the name and location for the copy that you are creating, and click OK.

Encapsulate Fields

Hiding your data and accessing it through an outward interface based on accessor methods is a good idea. Later you can change the data internals, preserving full compatibility with the code relied on the class and its available methods. The Encapsulate Fields refactoring allows you to hide your data and create the necessary accessors.

- Example
- Performing the Encapsulate Fields refactoring

Tip This refactoring is also available from UML Class diagram .

Example

Before After

```
//File Class.java
public class Class {
public String aString;
}
```

```
//File Class.java
public class Class {
private String aString;
public void setaString(String aString) {
this.aString = aString;
}
public String getaString() {
return aString;
}
}
```

```
//File AnotherClass.java
public class AnotherClass {
public Class aClass;
public void method() {
aClass.aString="string";
}
}
```

```
//File AnotherClass.java
public class AnotherClass {
public Class aClass;
public void method() {
aClass.setaString("string");
}
}
```

To perform the Encapsulate Fields refactoring, follow these steps

1. Select a class or a specific field within the class. The way you perform selection depends on the view where you do it.
   - In the editor: position the caret at the desired field or at any place inside the class to be refactored.
   - In the Project view: select the desired class.
   - In the Structure view: select one or more fields.
2. On the main menu or on the context menu of the selection, choose Refactor | Encapsulate Fields . The Encapsulate Fields dialog box is opened, displaying all fields, detected in the selected scope.
3. In the Fields to encapsulate area check the fields you want to create accessors for.
4. In the Encapsulate area, specify whether you want to create getter or setter methods, or both.
5. To replace all field occurrences with the calls to the appropriate accessor method, in the Options area check the option Use accessor even when field is accessible.
6. In the Encapsulated fields' visibility area specify the new visibility scope for the selected fields.
7. In the Accessors' visibility area select the visibility scope for the created accessor methods.
8. Preview and apply changes .

Extract Refactorings

In this section:

- Extract Delegate
- Extract Include File
- Extract Interface
- Extract Method
- Extract Method Object
- Extract Parameter Object
- Extract Superclass
- Extract Constant
- Extract Field
- Extract Functional Parameter
- Extract Parameter
- Extract Property
- Extract Variable

- The way extract constant, field, and variable refactorings are performed, depends on the setting of the Enable in-place refactoring check box in the Editor page of the Settings dialog.
- Extract refactorings are performed for the various expressions and blocks of code, including strings and substrings.
  Extract refactoring on substrings is supported for:
  - parts of strings with no formatting
  - parts of strings with concatenation via "+"
  - parts of strings with "%"-style formatting
  - parts of strings with "%()"-style formatting
  - new-style `"str.format()"` formatted strings

Extract Delegate

The Extract Delegate refactoring lets you extract some of the fields and methods of a class into a separate, newly created class. This refactoring is useful, when a class has grown too large and "does too many things". In such cases, it might be a good idea to split the class into smaller, more cohesive classes.

- Example
- Performing the Extract Delegate refactoring

Example

Before After

```
public class Foo
                    {
                    private String b;
                    public String getInfo() {
                    return ("(" + b + ")");
                    }
                    ...
                    }

                    public class Bar {
                    Foo foo;
                    String t2 = foo.getInfo();
                    ...
                    }
```

```
public class Foo
                    {
                    private final Info info = new Info();
                    public String getInfo() {
                    return info.getInfo();
                    }
                    ...
                    }

                    public class Info {
                    private String b;
                    public Info() {}
                    public String getInfo() {
                    return ("(" + b + ")");
                    }
                    }

                    public class Bar {
                    Foo foo;
                    String t2 = foo.getInfo();
                    ...
                    }
```

To perform Extract Delegate refactoring, follow these steps

1. Open the class in the editor, or select it in the Project tool window.
2. Select Refactor | Extract | Delegate from the main or the context menu.
3. In the Extract Class dialog that opens:
   - Specify the name and package for the class to be created.
   - Selects the fields and methods to be included in the new class.
   - Click Preview to see the usages of the selected fields or methods in the Find tool window . Select the usages to be included in the refactoring and click Do Refactor.

Note that this refactoring can also be accessed from a UML Class diagram .

Extract Include File

The Extract Include refactoring is used to extract a fragment of HTML, JSP/JSPX, JavaScript, or CSS code into a separate include file.

To extract an include file

1. In the editor, select the code block to be extracted and choose Refactor | Extract | Extract Include File on the main menu or on the context menu of the selection.

2. In the Extract Include File dialog box that opens, specify the name of the target include file in the Name for extracted include file text box.

   Tip Type the file name without an extension.

3. In the Extract to directory text box, specify the directory to store the include file in. Leave the predefined directory, or redefine it manually, or click the Browse button ... and choose the desired folder in the Select Target Directory dialog box that opens.

4. Click OK, when ready. IntelliJ IDEA extracts the selected source code into the specified file in the target directory and generates the corresponding reference in the source file.
   If there are any duplicates for the selected fragment, IntelliJ IDEA will suggest you to change them for the corresponding reference as well.

Extract interface

With the Extract Interface refactoring you have three options:

- Create an interface based on the methods of a class without applying the new interface immediately.
- Create an interface and apply it to the source code.
- Rename the original class, and it implements the newly created interface. In such case, IntelliJ IDEA changes all usages of the original class to use the interface where possible.

In addition, static final fields, declared in the initial class, can be moved to an interface. As a result, an interface will be created containing the specific methods and fields. Thereby, the specified class methods become implementations of the corresponding interface methods.

- Examples
- Extracting an interface

Examples

Here we have a class, and perform Extract Interface refactoring to create an interface based on the methods of the class.

Before **After**

```
// File AClass.java
class AClass {
public static final double
CONSTANT=3.14;
public void publicMethod()
{//some code here}
public void secretMethod()
{//some code here}
}
```

```
// File AClass.java
class AClass implements AnInterface
{
public void publicMethod()
{//some code here}
public void secretMethod()
{//some code here}
}
// File AnInterface.java
public interface AnInterface {
double CONSTANT=3.14;
void publicMethod();
}
```

Another example of the Extract Interface refactoring, when the Rename original class and use interface where possible option is selected.

Before **After**

```
public class FormerAClass
implements
AClass {
public void publicMethod()
{//some code here}
public void secretMethod()
{//some code here}
}
```

```
public interface AClass
{
double CONSTANT=3.14;
void publicMethod();
}
```

You can extract an interface from the class that already implements another interface. Let's extract interface from the class that implements `AnInterface`. Depending on whether we want `AnotherInterface` (extracted interface) to extend the `AnInterface` (existing one) or we want source `AClass` to implement them both, we will get the following code:

Extracted Interface extends the existing one:

```
class AClass implements
AnotherInterface
{
public void publicMethod()
{
//some code here
}
public void secretMethod()
{
//some code here
}
}
```

Extracted Interface:

```
public interface AnotherInterface
extends
AnInterface {
}
```

Source class implements both interfaces.

Source class:

```
class AClass implements
                    AnInterface, AnotherInterface {
                    public void publicMethod()
                    {
                    //some code here
                    }
                    public void secretMethod()
                    {
                    //some code here
                    }
                    }
```

Extracted Interface:

```
public interface AnotherInterface
                    {
                    }
```

Extracting an interface

1. Select a class in the Project view, Structure view, or place the caret anywhere within a class in the editor.
2. On the main menu or on the context menu of the selection, choose Refactor | Extract | Interface. The Extract Interface dialog box appears.
3. To extract a new interface, select the Extract Interface option and specify the name for the new interface.

   To rename the original class and make it an implementation of the newly created interface, select the Rename original class and use interface where possible option and specify the new name for the original class. IntelliJ IDEA will alter all original class usages to the usages of the implementing only where it is still necessary.
4. Specify the package, where the new interface will be located.
5. Select the class members you want to be listed in the interface in the Members to form interface area. The list shows all the methods of the class, as well as final static fields (constants).
6. In the JavaDoc area, select the action to be applied on the JavaDoc.
   - To leave it where it is, select the As is option.
   - To copy it to the extracted interface, select the Copy option.
   - To move it to the extracted interface, select the Move option.
7. Click Refactor to proceed.
8. Click Refactor when ready. If IntelliJ IDEA shows you a Refactoring Preview in the Find tool window, review the suggested changes. To have the interface extracted and the proposed changes applied, click Do Refactor.

Note If the class is used within the open project, IntelliJ IDEA proposes to replace the instances of the class with instances of the new interface. Note, that if an instance references a method or field, which are not defined in the interface, it won't be suggested for replacement.

Extract Method

- Basics
- Examples
  - Java example
  - JavaScript example
- Extracting a method
- Processing duplicates

Basics

When the Extract Method refactoring is invoked , IntelliJ IDEA analyses the selected block of code and detects variables that are the input for the selected code fragment and the variables that are output for it.

If there is exactly one output variable, it is used as a return value for the extracted method. In case there are multiple output variables, the Extract Method refactoring may not be applied, and the error message appears.

There are several workarounds to allow Extract Method work in this case. For example, you may introduce a special data-class that contains all output values.

Warning! The Extract Method refactoring has the following limitations:
- Refactoring does not work with multiple output values in automatic mode. You have to change your code before applying the refactoring.
- Refactoring does not work for a code fragment which conditionally returns from the containing method and is not placed at the end of it.

Examples

Java example

Before After

```
public void method() {
                    int a=1;
                    int b=2;
                    int c=a+b;
                    int d=a+c;
                    }
```

```
public void method() {
                    int a=1;
                    int b=2;
                    int c=add(a,b);
                    int d=add(a,c);
                    }
                    ...
                    private int add(int
                    a, int b)
                    {
                    return a+b;
                    }
```

```
public ArrayList method() {
                    String[] strings = {"a","b","c"};
                    ArrayList list = new ArrayList();
                    for (int i=0; i < strings.length; i++)
                    {list.add(strings[i]);}
                    return list;
                    }
```

```
public ArrayList method() {
                    String[] strings = {"a","ab","abc"};
                    ArrayList list=add(strings);
                    return list;
                    }

                    private ArrayList add(String[] strings) {
                    ArrayList list = new ArrayList();
                    for (int i=0; i < strings.length; i++)
                    {list.add(strings[i]);}
                    return list;
                    }
```

JavaScript example

Before After

```
function multiplication(a,b) {
                    c = a + b;
                    d = c * c;
                    return d;
                    }
```

```
function sum(a,b);
                    return a + b;

                    function multiplication(a,b) {
                    c = sum(a,b);
                    d = c * c;
                    return d;
                    }
```

Extracting a method

To extract a method, follow these steps

1. In the editor, select a block of code to be transformed into a method or a function.

   Tip The code fragment to form the method does not necessarily have to be a set of statements. It may also be an expression used somewhere in the code.

2. On the main menu or on the context menu of the selection, choose Refactor | Extract | Method or press `Ctrl+Alt+M`.

3. In the Extract Method dialog box that opens, specify the name of the new function.

4. To create a static method, select the Declare Static check box.

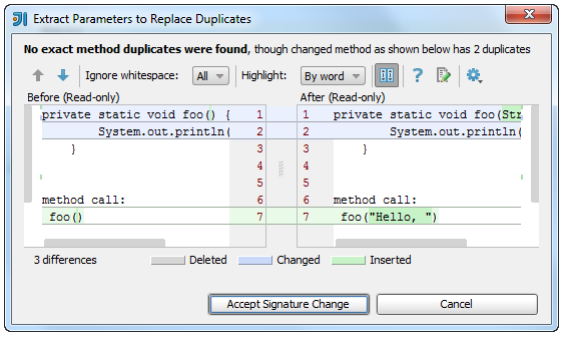
5. In the Parameters area, do the following:
   - Specify the variables to be passed as method parameters, by selecting/clearing the corresponding check boxes.
     If a parameter is disabled, a local variable of the corresponding type, with the initial value ... will be created in the extracted method, so that you will have to enter the initializer with an appropriate value manually.
   - Rename the desired parameters, by double-clicking the corresponding parameter lines and entering new names.

6. In the Visibility area define the method's visibility scope.

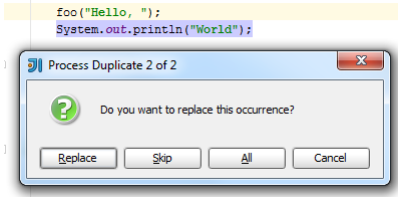
7. Check the result in the Signature Preview pane and click OK to create the method. The selected code fragment will be replaced with a method call.
   Additionally, IntelliJ IDEA will propose to replace any similar code fragments found within the current class.

Processing duplicates

IntelliJ IDEA detects the duplicated code fragments that may accept different values as parameters and shows the following suggestion in the format of the Defferences Viewer :



If you click the button Accept Signature Change, all the encountered duplicates will become highlighted, and IntelliJ IDEA will ask you for the confirmation:



Finally, after replacing the desired duplicates with the method call, you'll end up with the following code:

```
    foo("Hello, ");
    foo("World");

}

private static void foo(String x) {
    System.out.println(x);
}
```

You can also extract methods from the repetitive code fragments, which IntelliJ IDEA finds in course of the duplicates analysis . The encountered duplicates display in the Duplicates tool window , where you can try to replace them with method calls.

Extract Method Object

The Extract Method Object refactoring moves method into a new class, converting all the local variables to its fields, allowing you to decompose the method into other methods on the same object. It is an alternative to the Extract Method , and can be used when you have multiple return values in an extracted method.

Example

Before After

```
class Account {
                    int gamma (int val1, ...) {
                         //some computations
                                  return c-2*a;
                    }
                    }
```

```
class Account {
                    int gamma (int val1, ...) {
                    Calculations calculations = new
                    Calculations(val1, ...).invoke();
                    int c = calculations.getC();
                    int a = calculations.getA();
                    return c-2*a;
                    }
                    private class Calculations {
                    private int val1;
                    ...
                    private int a;
                    private int c;
                    public Calculations(int val1, ...) {
                    this.val1 = val1;
                    ...
                    }
                    public int getA() {return a;}
                    public int getC() {return c;}
                    public Calculations invoke()
                    {
                    ...//computations
                    return this;
                    }
                    }
                    }
```

To extract a method object, follow these steps

1. In the editor, select the method code block to be extracted into the object.
2. On the main menu, or from the context menu of the selection, choose Refactor | Extract | Method Object.
3. Select whether you want to create inner class, or anonymous class.

   Note You cannot extract the method object into an anonymous class, if the selected method code block contains local variables that should be accessed individually somewhere else. In this case method object can be extracted into inner class, that will contain needed getters.
4. If you want to create an inner class, you need to specify the name for the class and the visibility scope. You can also make the class static, if needed.
5. If you want to create an anonymous class, you should specify method's name.
6. In the Parameters area select the variables that will be used as a parameters.
7. Review Signature Preview and click OK.

Extract Parameter Object

The Extract Parameter Object refactoring allows you to select a set of parameters to a method, and either create a wrapper class for those parameters, or use an existing compatible wrapper class. All calls to the method selected will have their parameters appropriately wrapped, and all usages of the wrapped parameter will be replaced by the appropriate calls on the newly created parameter class.

Extracting a parameter object is useful if the number of parameters passed to a method has grown too large, or if the parameters have become complex enough to deserve first-class handling as their own class. Also, it is common to wrap primitive parameters as parameter objects, thus allowing interface and implementation to be decoupled as needed.

Example

Before **After**

```
public class A {
                    private void drawEdge(Graphics g, float edgeWidth,
                    int x1, int x2, int y1, int
                    y2) {
                    final Graphics2D g2d = (Graphics2D) g;
                    g2d.setStroke(new
                    BasicStroke(edgeWidth));
                    g.drawLine(x1, y1, x2, y2);
                    }
                    }
```

```
public class A {
                    private void drawEdge(Edge edge, Graphics g) {
                    final Graphics2D g2d = (Graphics2D) g;
                    g2d.setStroke(new
                    BasicStroke(edge.getEdgeWidth()));
                    g.drawLine(edge.getX1(), edge.getY1(), edge.getX2(), edge.getY2());
                    }
                    }

                    public class Edge {
                    private final float edgeWidth;
                    private final int x1;
                    ...
                    public Edge(float edgeWidth, int x1, int
                    x2, int y1,
                    int y2) {
                    this.edgeWidth = edgeWidth;
                    this.x1 = x1;
                    ...
                    }
                    public float getEdgeWidth()
                    {
                    return edgeWidth;
                    }
                    public int getX1() {
                    return x1;
                    }
                    ...
                    }
```

To extract a parameter object

1. Select the desired method. To do that, either open the class in question for editing, and position the caret at the method, click such method in the Structure view, or select it in the UML class diagram.
2. Choose Refactor | Extract | Parameter Object on the main menu or on the context menu of the selection.
3. In the Extract Parameter Object dialog box:
   - In the Parameter Class section, specify whether you want to create a new class, or use an existing one to wrap the parameters.
   - In the Parameters to extract list, check the parameters you want include in the new class.
   - Click Preview to make IntelliJ IDEA search for usages of the selected fields or methods, and display the refactoring preview results in the Find tool window. In the preview, you can include usages into refactoring or skip them. Click Do Refactor to apply refactoring to the selected usages.

     If you do not want to view usages, click Refactor. In this case, the usages will be changed immediately.

   The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Extract Superclass

- Basics
- Example
- Extracting a superclass

Basics

The Extract Superclass refactoring has two options:

- Create a superclass based on an existing class.
- Rename the original class so that it becomes an implementation for the newly created superclass. In this case, IntelliJ IDEA changes all original class usages to use a superclass where possible.

Fields and methods in the original class can be moved to the superclass. Also for a method, you can transfer only the method declaration but not the implementation declaring the method as abstract in the superclass. As a result, you will have the superclass and the original class inherited from the superclass.

Example

Before **After**

```
// File Class.java
public class Class {
public int varInt;
private double varDouble;
public static final int CONSTANT = 0;
public void publicMethod()
{
...
}
public void hiddenMethod()
{
...
}
public void setVarDouble(double var) {
this.varDouble = var;
}
public double getVarDouble()
{
return
varDouble;
}
}
```

```
// File Class.java
public class Class extends
SuperClass {
public int varInt;
public void publicMethod()
{
...
}
public void hiddenMethod()
{
...
}
}

// NEW file SuperClass.java
public abstract class SuperClass
{
private double varDouble;
public static final int CONSTANT = 0;
public abstract void publicMethod();
public void setVarDouble(double var) {
this.varDouble = var;
}
public double getVarDouble()
{
return varDouble;
}
}
```

To extract a superclass

1. Select the desired class in one of the views, or just open it in the editor.
2. On the on the main menu or on the context menu, choose Refactor | Extract | Superclass.
3. In the Extract Superclass dialog box:
   - To create a new superclass, select the Extract superclass option and specify the name for the new superclass.
     To rename the original class and make it an implementation of the newly created superclass, select the Rename original class and use superclass where possible option and specify the new name for the original class. IntelliJ IDEA will change all original class usages to the usages of the implementing class only where it is still necessary.
   - Specify the package, where the new superclass will be located.
   - In the Members to Form Superclass area, select the class members you want to be moved or delegated to the superclass.
     To leave the method implementation within the current class, and declare it abstract in the extracted superclass, select the Make Abstract check box.
   - In the JavaDoc area, select the action to be applied on the JavaDoc.
     - To leave it where it is, select the As is option.
     - To copy it to the extracted superclass, select the Copy option.
     - To move it to the extracted superclass, select the Move option.
4. Click Refactor to proceed.
   If the class is used within the current project, IntelliJ IDEA proposes to replace the instances of this class with the instances of the new superclass. If a class instance references a member, which is not defined in the superclass, it won't be suggested for replacement.

Extract Constant

The Extract Constant refactoring is a special case of the Extract Field refactoring intended to provide a fast and convenient way to create final static fields.

On this page:

– Example
– Extracting a Java constant in-place
– Extracting a constant using the dialog box

Example

Before After

```
public class Class
                    {
                    public void method() {
                    String string = "string";
                    ArrayList list = new ArrayList();
                    list.add(string);
                    anotherMethod(string);
                    ...
                    }
                    }
```

```
public class Class
                    {
                    @NonNls
                    private static final String
                    STRING ="string";
                    public void method() {
                    ArrayList list = new ArrayList();
                    list.add(STRING);
                    anotherMethod(STRING);
                    ...
                    }
                    }
```

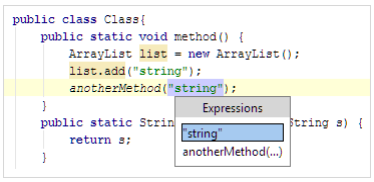
```
public class Class
                    {
                    public void method() {
                    ArrayList list = new ArrayList();
                    list.add("string");
                    anotherMethod("string");
                    ...
                    }
                    }
```

```
public class Class
                    {
                    @NonNls
                    private static final String
                    STRING ="string";
                    public void method() {
                    ArrayList list = new ArrayList();
                    list.add(STRING);
                    anotherMethod(STRING);
                    ...
                    }
                    }
```

To extract a Java constant in-place

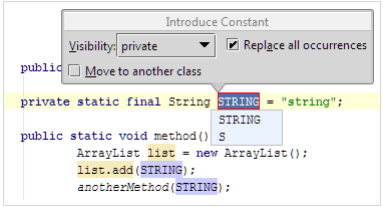
The in-place refactorings are enabled in IntelliJ IDEA by default. So, if you haven't changed this setting, the Extract Constant refactoring for Java is performed in-place, right in the editor:

1. Place the cursor within the expression or declaration of a variable to be replaced by a constant.
2. Do one of the following:
   – Press `Ctrl+Alt+C`.
   – Choose Refactor | Extract Constant on the main menu or on the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



4. If more than one occurrence of the expression is found within the class, specify whether you wish to replace only the selected occurrence, or all the found occurrences with the new constant.
5. If you want the constant to be defined in a different class, select the Move to another class check box.
6. If necessary, change the type of the new constant.
   To move to the type, press `Shift+Tab`. Then, select the required type from the list, or edit the type in the box with the red border.

   Now, to move back to the constant name, press `Tab`.
7. Specify the name of the constant. Select the name from the list or type the name in the box with a red border.



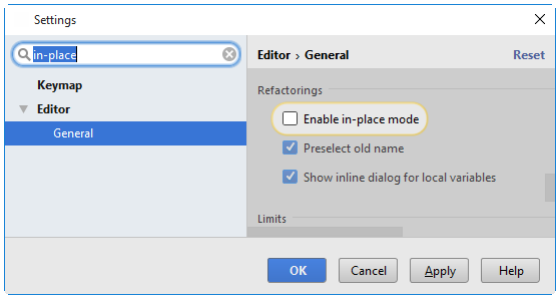
8. To complete the refactoring, press `Tab` or `Enter`.

If you haven't completed the refactoring and want to cancel the changes you have made, press `Escape`.

9. If you have selected to move the constant definition to a different class, specify the associated settings in the Move Members dialog .

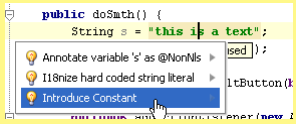
To extract a constant using the dialog box

If the Enable in-place refactorings check box is cleared on the Editor settings, the Extract Constant refactoring is performed by means of the Extract Constant Dialog dialog box.



1. If you are working with Java code, make sure that the Enable in place refactorings option is off in the editor settings. (By default, the Extract Constant refactorings for Java are performed in-place .)
2. In the editor, select the expression or variable to be replaced with a constant, or just place the cursor within such an expression or variable declaration.
3. In the main or the context menu, choose Refactor | Extract Constant, or press `Ctrl+Alt+C`.
4. In the Expressions pop-up menu, select the expression to be replaced. Note that IntelliJ IDEA highlights the selected expression in the editor.
5. In the Extract Constant Dialog dialog that opens:
   1. Specify the name of the new constant.
   2. Select the class where the constant will be introduced. If you select an `enum` class here, use the Introduce as enum constant option to specify whether the constant should be an `enum` constant, or a usual field.
   3. In the Visibility area, select the visibility scope for the new constant.
   4. If the new constant is going to replace an existing variable, you can choose to delete the corresponding variable declaration. To do that, use the Delete variable declaration check box.
   5. To replace all the occurrences of the selected expression (if the selected expression is found more than once in the class), select the Replace all occurrences check box.
   6. In the projects configured for using annotations, you can annotate the constant of the String type as `@NonNls` to prevent it from change during possible localizations. To do so, select the option Annotate field as @NonNls.
   7. Click OK.

Tip You can also invoke the refactoring from the intention action drop-down menu:



Extract Field

The Extract Field refactoring declares a new field and initializes it with the selected expression. The original expression is replaced with the usage of the field.

– Example
– Extracting a field in-place
– Extracting a field using the dialog box

Example

Before **After**

```
public class Class
                    {
                    AnotherClass anotherClass;
                    public void method() {
                    int a = 1;
                    ...
                    int b = a +
                    anotherClass.intValue();
                    int c = b +
                    anotherClass.intValue();
                    }
                    }
```

```
public class Class
                    {
                    public AnotherClass anotherClass;
                    private int number;
                    public Class() {
                    number = anotherClass.intValue();
                    }
                    public void method() {
                    int a = 1;
                    ...
                    int b = a +
                    number;
                    int c = b +
                    number;
                    }
                    }
```

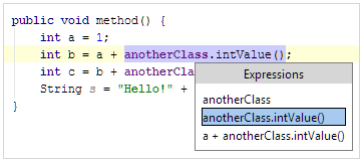
```
public class Class
                    {
                    AnotherClass anotherClass;

                    public void method() {
                    int a = anotherClass.innerClass.i;
                    int b = anotherClass.innerClass.j;
                    }
                    }
```

```
public class Class
                    {
                    public AnotherClass anotherClass;
                    private AnotherClass.InnerClass
                    aClass = anotherClass.innerClass;
                    public void method() {
                    int a = aClass.i;
                    int b = aClass.j;
                    }
                    }
```

To extract a field in-place

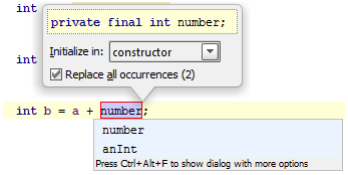
The in-place refactorings are enabled in IntelliJ IDEA by default. So, if you haven't changed this setting, the Introduce Field refactorings are performed in-place, right in the editor:

1. Place the cursor within the expression or declaration of a variable to be replaced by a field.
2. Do one of the following:
   – Press `Ctrl+Alt+F`.
   – Choose Refactor | Introduce Field on the main menu, or on the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



4. If more than one occurrence of the expression is found within the class, specify whether you wish to replace only the selected occurrence, or all the found occurrences with the new constant:
5. If necessary, change the type of the new field.
   To move to the type, press `Shift+Tab`. Then, select the required type from the list, or edit the type in the box with the read border.

   Now, to move back to the field name, press `Tab`.
6. If relevant, specify where the new field will be initialized - in the current method, or in a class constructor.
7. Specify the name of the field. Select the name from the list or type the name in the box with a red border.



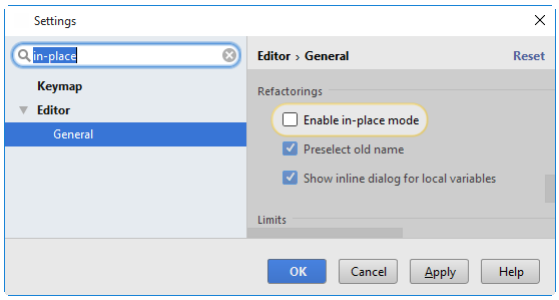
8. To complete the refactoring, press `Tab` or `Enter`.
   If you haven't completed the refactoring and want to cancel the changes you have made, press `Escape`.

Note that sometimes you may need to press the corresponding key more than once.

```
public class Class {
    AnotherClass anotherClass;
    private int number;

    public Class() {
        number = anotherClass.intValue();
    }

    public void method() {
        int a = 1;
        int b = a + number;
        int c = b + number;
```

To extract a field using the dialog box

If the Enable in place refactorings check box is cleared on the Editor settings, the Extract Field refactoring is performed by means of the Introduce Field .



1. In the editor, select the expression or variable to be replaced with a field, or just place the cursor within such an expression or variable declaration.
2. In the main menu, or the context menu of the selection, choose Refactor | Extract Field, or press `Ctrl+Alt+F`.
3. In the Expressions pop-up menu, select the expression to be replaced. Note that IntelliJ IDEA highlights the selected expression in the editor.
4. In the Extract Field Dialog dialog that opens:
    1. Select the type of the new field from the Field of type list.
    2. Specify the name of the field.
    3. Specify where the new field should be initialized by selecting the necessary option under Initialize in.
    4. In the Visibility area, select the visibility scope for the new field.
    5. If you want the new field to be declared `final`, select the Declare final check box.
    6. If the new field is going to replace an existing variable, you can choose to delete the corresponding variable declaration. To do that, use the Delete variable declaration check box.
    7. To replace all the occurrences of the selected expression (if the selected expression is found more than once in the class), select the Replace all occurrences check box.
    8. Click OK.

Extract Functional Parameter

When you perform the Extract Functional Parameter refactoring, IntelliJ IDEA:

1. Analyzes the selected code fragment to find out what the method signature would be if you extracted this fragment into a new separate method.
2. Finds all functional interfaces with this method signature and suggests that you select one of them. (Only the interfaces marked with `@FunctionalInterface` or ones belonging to well-known libraries such as Guava, Apache Collections, etc. are suggested.)
3. Wraps the code fragment with an anonymous class based on the selected interface and uses this anonymous class as a parameter.
4. Where appropriate, makes related code adjustments.

– Example
– Performing the Extract Functional Parameter refactoring

Example

Before **After**

```
@FunctionalInterface
                    public interface Person {
                    public void sayHello (String s);
                    }

                    public class Hello {
                    private void printHello ()
                    {
                    String s="Hello";
                    System.out.println(s);
                    }
                    private void printText () {
                    printHello();




                    }
                    }
```

```
@FunctionalInterface
                    public interface Person {
                    public void sayHello (String s);
                    }

                    public class Hello {
                    private void printHello(Person person) {
                    String s = "Hello";
                    person.sayHello(s);
                    }
                    private void printText () {
                    printHello(new Person()
                    {
                    public void sayHello(String s) {
                    System.out.println(s);
                    }
                    });
                    }
                    }
```

In this example, the refactoring is performed on `System.out.println(s);` within `Hello.printHello()`.

IntelliJ IDEA finds all functional interfaces with the appropriate method signature (`(String) -> void`) and suggests that you select one of them. (In this example, the interface `Person` is selected.)

As a result:

– The selected fragment (`System.out.println(s);`) is wrapped with an anonymous class based on the interface `Person`. This anonymous class is passed as a parameter to the call to `printHello()` within `printText()`.
– `Person` is added as a parameter to `printHello()` and the initially selected code fragment is replaced with the call to the method of the interface (`sayHello()`).

Performing the Extract Functional Parameter refactoring

1. Select the code fragment of interest and do one of the following:
   – Press `Ctrl+Shift+Alt+P`.
   – Select Refactor | Extract | Functional Parameter from the main or the context menu.
   – Select Refactor | Refactor This from the main menu (`Ctrl+Shift+Alt+T`), and select Functional Parameter.
2. Select the desired functional interface from the list.
3. Specify the refactoring options in the Extract Parameter dialog .

Extract Parameter

The Extract Parameter refactoring is used to add a new parameter to a method declaration and to update the method calls accordingly.

– Extract Parameter in Java

See also Change Signature .

Extract Parameter in Java

This section discusses the Extract Parameter refactoring in Java.

- Examples
- Extracting a parameter in Java in-place
- Extracting a parameter in Java using the Extract Parameter dialog
- Special notes
- Side effects

Examples

When extracting a new parameter to a method, the following two general approaches may be used depending on how the existing method calls should be handled:

- If it's possible to change all the existing method calls, a new parameter may be added to an existing method. The method calls in this case are changed accordingly, see the first of the examples .
- If the existing method calls cannot be changed, a method with the existing signature is kept. The new parameter in this case is defined in a new, overloading method, see the second of the examples .

In this example, the string value `"Hello, World!"` in the method `generateText()` is replaced with the new parameter `text`. The value `"Hello, World!"` is passed to the method in the updated method call `generateText("Hello, World!")`.

Before **After**

```
public class HelloWorldPrinter
                    {
                    public static void print() {
                    System.out.println(generateText());
                    }
                    private static String generateText() {
                    return "Hello, World!".toUpperCase();
                    }
                    }
```

```
public class HelloWorldPrinter
                    {
                    public static void print() {
                    System.out.println(generateText("Hello, World!"));
                    }
                    private static String generateText(String text) {
                    return text.toUpperCase();
                    }
                    }
```

In this example a new overloading method is created and the new parameter is extracted in the definition of this method (the second of the `generateText()` methods). The signature of the existing `generateText()` method is not changed. However, the method itself has been modified. Now, it calls the new `generateText()` method and passes the value `"Hello, World!"` to it in this call. Note that the existing call of `generateText()` (in the method `print()`) is not changed.

In IntelliJ IDEA, this way of extracting a parameter corresponds to the option Delegate via overloading method.

Before **After**

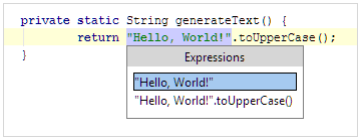
```
public class HelloWorldPrinter
                    {
                    public static void print() {
                    System.out.println(generateText());
                    }
                    private static String generateText() {
                    return "Hello, World!".toUpperCase();
                    }
                    }
```

```
public class HelloWorldPrinter
                    {
                    public static void print() {
                    System.out.println(generateText());
                    }
                    private static String generateText() {
                    return generateText("Hello, World!");
                    }
                    private static String generateText(String text) {
                    return text.toUpperCase();
                    }
                    }
```

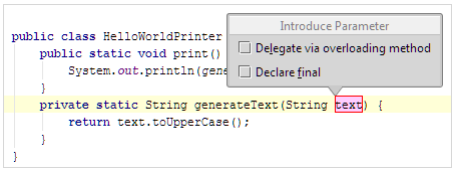
Extracting a parameter in Java in-place

The in-place refactorings are enabled in IntelliJ IDEA by default. So, if you haven't changed this setting, the Extract Parameter refactorings for Java are performed in-place, right in the editor:

1. In the editor, place the cursor within the expression to be replaced by a parameter.
2. Do one of the following:
   - Press `Ctrl+Alt+P`.
   - Choose Refactor | Extract | Parameter in the main menu.
   - Choose Refactor | Extract | Parameter from the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



4. Type the parameter name in the box with a red border.



5. Select the necessary options in the Extract Parameter option box.
   - If more than one occurrence of the expression is found within the method body, you can choose to replace only the selected occurrence or all the found occurrences with the references to the new parameter. Use the Replace all occurrences check box to specify your intention.
   - If you don't want to change the existing method calls, select the Delegate via overloading method check box. For more information on how this option works, see Examples .
   - To declare the parameter `final`, select the Declare final check box.

6. If necessary, change the type of the new parameter. To do that, press `Shift+Tab` and edit the type in the box with the read border. (If, at this step, you want to return to editing the parameter name, press `Tab`.)

7. To complete the refactoring, press `Tab` or `Enter`.
   If you haven't completed the refactoring and want to cancel the changes you have made, press `Escape`.

   Note that sometimes you may need to press the corresponding key more than once.

Extracting a parameter in Java using the Extract Parameter dialog

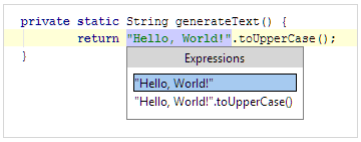
To be able to use the Extract Parameter dialog (instead of performing the refactoring in-place), make sure that the Enable in place refactorings option is off in the editor settings.

Once this is the case, you perform the Extract Parameter refactoring as follows:

1. In the editor, place the cursor within the expression to be replaced by a parameter.

2. Do one of the following:
   - Press `Ctrl+Alt+P`.
   - Choose Refactor | Extract | Parameter in the main menu.
   - Choose Refactor | Extract | Parameter from the context menu.

3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



4. In the Extract Parameter dialog :
   1. Usually, IntelliJ IDEA sets a proper parameter type itself. If necessary, you can select another appropriate type from the Parameter of type list.
   2. Specify the parameter name in the Name field.
   3. If more than one occurrence of the expression is found within the method body, you can choose to replace only the selected occurrence or all the found occurrences with the references to the new parameter. Use the Replace all occurrences check box to specify your intention.
   4. To declare the parameter `final`, select the Declare final check box.
   5. If the expression contains a direct call to a class field that has a getter, specify the getter-related options.
   6. If the expression contains local variables, specify how these local variables will be treated in the corrected method calls.

5. Preview and apply changes .

Special notes

The following specific issues should be mentioned:

- If you want a field to be provided as a new parameter in the method declaration, in a method call this field will be presented as a field of a class instance.
- If a class member is inaccessible (for instance, in the example above the field is private), it will be inserted in a method call but will be highlighted as an error making this file uncompilable.
- If you use for the Extract Parameter refactoring a field with a getter, you will be prompted with an extended dialog. The dialog has an option group Replace fields used in expressions with their getters :
  - Do not replace: None of the fields will be replaced with calls to the getter.
  - Replace fields inaccessible in usage context: Only the fields that cannot be directly accessed from the usage context will be replaced with calls to the getter.
  - Replace all fields: All fields will be replaced with calls to the getter.
- Applying the refactoring on a local variable will call the Extract Parameter dialog box with additional check boxes:
  - Replace all occurrences (<number_of_occurrences> occurrences): If enabled, all occurrences of the selected variable will be replaced with a parameter and the Delete variable definition check box is enabled. Otherwise, only the selected variable usage will be replaced with a parameter.
  - Delete variable definition: If enabled, the variable definition will be deleted.
  - Use variable initializer to initialize parameter: If enabled, the variable initializer will be used to initialize the parameter in the method call.

Side effects

Using the Extract Parameter refactoring can have unexpected side effects if applied on class instances or expressions which are actual method parameters. For instance,

in case of such code:

```
class AClass {
                      int field;
                      int method() {
                      return field;
                      }
                      }

                      class Usage {
                      void method(List list) {
                      int sum = 0;
                      for(Iterator it =
                      list.iterator(); it.hasNext(); ) {
                      sum += ((AClass)
                      it.next()).method();
                      }
                      }
                      }
```

The code after refactoring applied to the field `field`:

```
class AClass {
                      int field;
                      int method(int newfield) {
                      return newfield;
                      }
                      }

                      class Usage {
                      void method(List list) {
                      int sum = 0;
                      for(Iterator it =
                      list.iterator; it.hasNext(); ) {
                      sum += ((AClass)
                      it.next()).method(((AClass) it.next()).field);
                      }
                      }
                      }
```

The iterator value is increased twice which is, actually, not the behavior you would expect.

However, IntelliJ IDEA can use a temporary variable successfully and resolve such cases as increment/decrement/assignment operations and the new keyword usage. For instance:

```
public int myMethod(List
                      list) {
                      return list.size();
                      }

                      public void anotherMethod()
                      {
                      myMethod(new ArrayList());
                      }
```

Same code after refactoring looks as follows:

```
public int myMethod(List
                      list, int newPar)
                      {
                      return list.size();
                      }
                      public void anotherMethod()
                      {
                      final ArrayList list = new ArrayList();
                      myMethod(list, list.size());
                      }
```

The new variable `list` was created and all parameters used for the method call are provided using this variable.

Extract Property

In Maven projects , while editing `pom.xml` , one needs to define a property and replace the occurrences of some value - artifact version, for example, - with this property.

For these purposes, IntelliJ IDEA provides the Extract Property refactoring.

Extract Property refactoring creates a new property definition in the specified `pom.xml` file, finds all the occurrences of the selected string in the hierarchy of `pom.xml` files, and replaces them with the above property in the format: `${<property_name>}`

The Extract Property refactoring lets you move expressions and local declarations to properties.

Examples

Before After

In this example, artifact version is replaced with a property, which is declared in the same `pom.xml` file:

```
<artifactId>submodule<artifactId>
```

```
<artifactId>${submodule}</artifactId>
<properties>
<submodule>submodule</submodule>
</properties>
```

In this example, artifact version in a `pom.xml` file is replaced with a property, which is declared in its parent `pom.xml` file:

```
<parent>
<artifactId>HelloWorld</artifactId>
<version>1.0</version>
</parent>
<artifactId>submodule<artifactId>
```

Parent:

```
<artifactId>HelloWorld</artifactId>
...
<modules>
<module>${submodule}</module>
</modules>
...
<properties>
<submodule>submodule</submodule>
</properties>
```

Child:

```
<parent>
<artifactId>HelloWorld</artifactId>
<version>1.0</version>
</parent>
<artifactId>${submodule}</artifactId>
```

To extract a property in a pom.XML file

1. Open the desired `pom.xml` file for editing, and place the caret somewhere inside the value you want to replace.

   Tip You can also select a certain substring; in this case the refactoring will apply to the selection.

2. Press `Ctrl+Alt+V` , or choose Refactor | Extract | Property on the context menu, or on the main menu. Note that selection will be automatically expanded up to the enclosing tags.

3. In the Extract Property dialog box, do the following:
   - In the Name field, specify the name you want to assign to the new property. IntelliJ IDEA suggests a number of suitable names. You can select one from the drop-down list, or type the desired name manually.
   - In the Project drop-down list, select the project where the new property will be declared.

   Click OK.

4. The subsequent workflow depends on the specific `pom.xml` file the refactoring has been invoked from.
   - The occurrence, for which the refactoring has been invoked, will be replaced silently. If the string occurs several times, the replace usage dialog will be displayed for these occurrences. You have to specify whether you want to replace each occurrence.
   - If the `pom.xml` is a parent file and the new property will be declared in it, IntelliJ IDEA replaces the values in the parent file silently, and displays the other occurrences in the Find tool window . You have to specify whether you want to replace each occurrence.
   - If the `pom.xml` is inherited from a parent file, IntelliJ IDEA replaces the occurrence in the child file silently. This change is not propagated to the parent `pom.xml` .

   In each case, IntelliJ IDEA adds new property declaration to the specified `pom.xml` file.

Extract Variable

– Basics
– Java Examples
– Extracting variable in-place
– Extracting variable with a dialog

Basics

The Extract Variable refactoring puts the result of the selected expression into a variable. It declares a new variable and uses the expression as an initializer. The original expression is replaced with the new variable (see the examples below).

In Java, the type of the new variable corresponds to that returned by the expression. There is an option of declaring the new variable as `final` .

To perform this refactoring, you can use:

– In-place refactoring . In this case you specify the new name right in the editor.
– Refactoring dialog , where you specify all the required information. To make such a dialog accessible, you have to clear the check box Enable in-place mode in the editor settings.

You can select the expression to be replaced with a variable yourself. You can as well use smart expression selection . In this case IntelliJ IDEA will help you select the desired expression.

This refactoring is also available for JavaScript and Sass .

Java Examples

Before After

```
public void method() {
                    int a = 1;
                    ...
                    int b = a +
                    anotherClass.intValue();
                    int c = b +
                    anotherClass.intValue();
                    }
```

```
public void method() {
                    int a = 1;
                    ...
                    int number = anotherClass.intValue();
                    int b = a +
                    number;
                    int c = b +
                    number;
                    }
```

```
public void method() {
                    int a = anotherClass.innerClass.i;
                    int b = anotherClass.innerClass.j;
                    }
```

```
public void method() {
                    AnotherClass.InnerClass aClass
                    = anotherClass.innerClass;
                    int a = aClass.i;
                    int b = aClass.j;
                    }
```

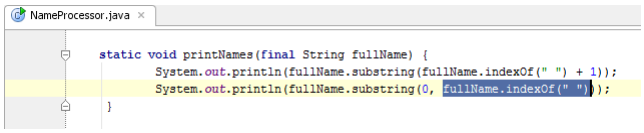
```
static void printNames(final String fullName) {
                    System.out.println(fullName.substring(fullName.indexOf("
                    ") + 1));
                    System.out.println(fullName.substring(0, fullName.indexOf(" ")));
                    }
```

```
static void printNames(final String fullName) {
                    int firstNameEndIndex = fullName.indexOf(" ");
                    System.out.println(fullName.substring(firstNameEndIndex + 1));
                    System.out.println(fullName.substring(0, firstNameEndIndex));
                    }
```

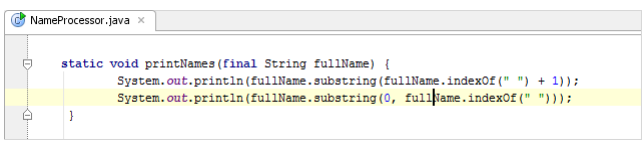
Extracting variable in-place

To extract a variable using in-place refactoring, follow these steps

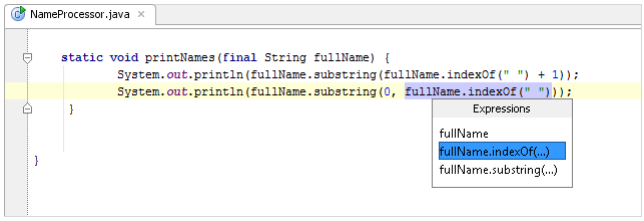
1. In the editor, select the expression to be replaced with a variable. You can do that yourself or use the smart expression selection feature to let IntelliJ IDEA help you. So, do one of the following:
   – Highlight the expression. Then choose Refactor | Extract | Variable on the main menu or on the context menu.
   Alternatively, press `Ctrl+Alt+V` .



   – Place the cursor before or within the expression. Choose Refactor | Extract Variable on the main menu or on the context menu. or press `Ctrl+Alt+V` .



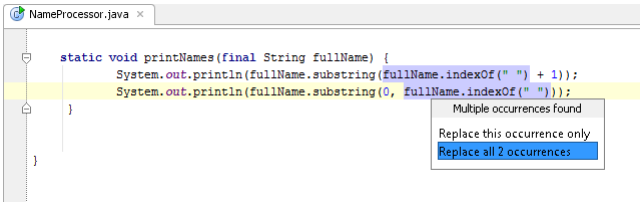
   In the Expressions pop-up menu, select the expression. To do that, click the required expression. Alternatively, use the Up and Down arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



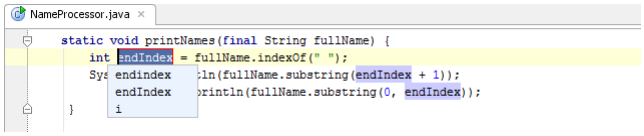
Note The Expressions pop-up menu contains all the expressions appropriate for the current cursor position in the editor.

When you navigate through the suggested expressions in the pop-up, the code highlighting in the editor changes accordingly.

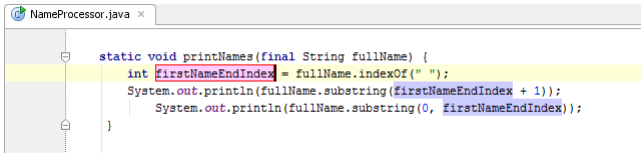
2. If more than one occurrence of the selected expression is found, select Replace this occurrence only or Replace all occurrences in the Multiple occurrences found pop-up menu. To select the required option, just click it. Alternatively, use the Up and Down arrow keys to navigate to the option of interest, and press `Enter` to select it.



3. Specify the name of the variable. Do one of the following:
   - Select one of the suggested names from the pop-up list. To do that, double-click the suitable name. Alternatively, use the Up and Down arrow keys to navigate to the name of interest, and `Enter` to select it.



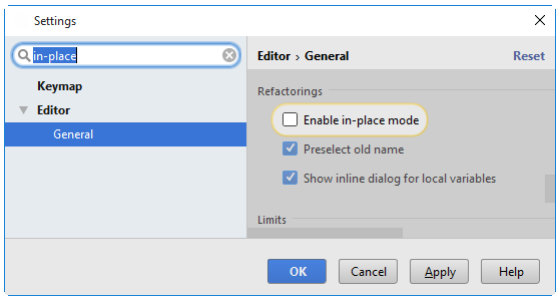
   - Edit the name by typing. The name is shown in the box with red borders and changes as you type. When finished, press `Enter`.



Extracting variable with a dialog

To extract a variable using the dialog box

If the Enable in place refactorings check box is cleared in the Editor settings, the Extract Variable refactoring is performed by means of the dialog box.



1. In the editor, select the expression to be replaced with a variable. You can do that yourself or use the smart expression selection feature to let IntelliJ IDEA help you. So, do one of the following:
   - Highlight the expression. Then choose Refactor | Extract | Variable on the main menu or on the context menu.
     Alternatively, press `Ctrl+Alt+V`.
   - Place the cursor before or within the expression. Choose Refactor | Extract Variable on the main menu or on the context menu. or press `Ctrl+Alt+V`.
     In the Expressions pop-up menu, select the expression. To do that, click the required expression. Alternatively, use the Up and Down arrow keys to navigate to the expression of interest, and then press `Enter` to select it.

Note The Expressions pop-up menu contains all the expressions appropriate for the current cursor position in the editor.

When you navigate through the suggested expressions in the pop-up, the code highlighting in the editor changes accordingly.

2. In the Extract Variable Dialog dialog:
   1. Specify the variable name next to Name. You can select one of the suggested names from the list or type the name in the Name box.
   2. If more than one occurrence of the selected expression is found, you can select to replace all the found occurrences by selecting the corresponding check box. If you want to replace only the current occurrence, clear the Replace all occurrences check box.
   3. If you want to declare the new variable `final`, select the Declare final check box. (This option is available only for Java.)
   4. For ActionScript, you can choose to introduce a constant rather than a variable. To do that, select the Make constant check box.
      Note that ActionScript is not supported in IntelliJ IDEA Community Edition.
   5. Click OK.

Refer to the following sections for the other language- and framework-specific refactorings:

- For the JavaScript procedure, refer to the section Extract Variable in JavaScript .
- For the Sass procedure, refer to the section Extract Variable for Sass .

Generify Refactoring

The Generify refactoring is designed to transform existing code that does not use Generics, into the Generics-aware code. The refactoring analyzes existing code, and for each raw type creates safe and consistent parameter type.

IntelliJ IDEA tries to generate code, which is as correct as possible from the Java point of view. In other words, each context introduces some type restrictions, and the refactoring produces the best possible type, that does not contradict to the existing contexts.

Example

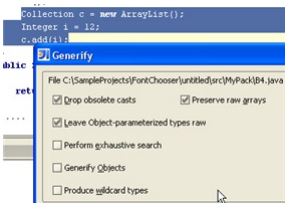
Before **After**

```
public void method() {
                    List list = new LinkedList();
                    list.add("string");
                    }
```

```
public void method() {
                    List<String> list
                    = new LinkedList<String>();
                    list.add("string");
                    }
```

To generify

1. Select the level of code transformation, which can be a method, a class, a package or directory, in the Project or Structure view, or place the cursor on the class or method name in the editor. If you want to apply generics to a single code fragment, just select it in the editor.
2. On the main menu, or on the context menu of the selection, choose Refactor | Generify . The Generify dialog box appears.



3. Define the refactoring options. Refer to the dialog description for details.
4. Preview and apply changes .

Inline

IntelliJ IDEA provides the following inline refactorings:

- Inline Variable refactoring replaces redundant variable usage with its initializer. See example . This refactoring is opposite to the Extract Variable refactoring.

  Tip The variable must be initialized at declaration. If the initial value is modified somewhere in the code, only the occurrences before modification will be inlined.

- Inline Method refactoring results in placing method's body into the body of its caller(s). You can opt to:
  - inline all occurrences of the method, and delete the method
  - inline only a single occurrence, and retain the method

  See example . This refactoring is opposite to the Extract Method refactoring.
- Inline to Anonymous Class refactoring allows replacing redundant class with its contents.
- Inline Constructor allows compressing a chain of constructors, if one of them is a special case of another. See example .
- Inline JSP/JSPX works like a regular Java inline refactoring.
- Inline Superclass refactoring results in pushing superclass' methods into the class where they are used, and removing the superclass.

Examples

Inline Variable

BeforeAfter

```
public void method() {
                    int number = anotherClass.intValue();
                    int b = a +
                    number;
                    }
```

```
public void method() {
                    int b = a +
                    anotherClass.intValue();
                    }
```

```
public void method() {
                    AnotherClass.InnerClass aClass
                    = anotherClass.innerClass;
                    int a = aClass.i;
                    }
```

```
public void method() {
                    int a = anotherClass.innerClass.i;
                    }
```

Inline Method

BeforeAfter

```
public void method() {
                    int c=add(a,b);
                    int d=add(a,c);
                    }

                    private int add(int
                    a, int b)
                    {
                    return a+b;
                    }
```

```
public void method() {
                    int c= a +
                    b;
                    int d= a +
                    c;
                    }
```

```
public ArrayList method() {
                    String[] strings = {"a","b","c"};
                    ArrayList list=add(strings);
                    return list;
                    }

                    private ArrayList add(String[] strings) {
                    ArrayList list = new ArrayList();
                    for (int i=0; i< strings.length; i++)
                    {list.add(strings[i]);}
                    return list;
                    }
```

```
public ArrayList method() {
                    String[] strings = {"a","ab","abc"};
                    ArrayList list1 = new ArrayList();
                    for (int i=0; i< strings.length; i++)
                    {list.add(strings[i]);}
                    ArrayList list = list1;
                    return list;
                    }
```

Inline Constructor

BeforeAfter

```
public class Class
                    {
                    public int varInt;
                    public Class(int i) {
                    varInt=i;
                    }
                    public void method() {
                    Class aClass=new Class(0);
                    ...
                    }
                    }
```

```
public class Class
                    {
                    public int varInt;
                    public Class() {
                    this(0);
                    }

                    public Class(int i) {
                    varInt=i;
                    }

                    public void method() {
                    Class aClass=new Class();
                    ...
                    }
                    }
```

Inline Superclass

Before **After**

```
public class Bar
                    {
                    ...
                    int calculations1() { ... }
                    int calculations2() { ... }
                    }

                    class Foo extends Bar {
                    int someMethod() {
                    ...
                    if (something > calculations1()) {
                    ...
                    return calculations2();
                    }
                    ...
                    }
                    }
```

```
class Foo {
                    ...
                    int someMethod() {
                    ...
                    if (something > calculations1()) {
                    ...
                    return calculations2();
                    }
                    ...
                    }
                    int calculations1() {...}
                    int calculations2() {...}
                    }
```

To perform the inline refactoring

1. Place the caret in the editor at the desired symbol to be inlined.
2. Do one of the following:
   - On the main menu or on the context menu, choose Refactor | Inline.
   - Press `Ctrl+Alt+N`.
3. In the Inline dialog , that corresponds to the selected symbol, specify the inlining options.
4. Preview and apply changes .

Invert Boolean

The Invert Boolean refactoring allows you to change the sense of a Boolean method or variable to the opposite one.

Tip This refactoring is also available from UML Class diagram .

Example

Before **After**

```
private double a;
                    ...
                    public boolean method() {
                    if (a > 15
                    && a < 100)
                    {
                    a = 5;
                    return true;
                    }
                    return false;
                    }
```

```
private double a;
                    ...
                    public boolean method() {
                    if (a > 15
                    && a < 100)
                    {
                    a = 5;
                    return false;
                    }
                    return true;
                    }
```

```
boolean b = true;
                    ...
                    public double method() {
                    ...
                    b = false;
                    ...
                    }
```

```
boolean b = false;
                    ...
                    public double method() {
                    ...
                    b = true;
                    ...
                    }
```

To invert the sense of a Boolean method or variable

1. Place the caret at the name of the method or variable to be refactored.
2. Do one of the following:
   - On the main menu choose Refactor | Invert Boolean.
   - On the context menu, choose Refactor | Invert Boolean.
3. In the Invert Boolean dialog, specify the name for the inverted method or variable.
4. Preview and apply changes .

Make Class Static

The Make Class Static refactoring allows you to convert an inner class into a static one, and automatically corrects all references to the class in the code.

To make a class static

1. In the editor, place the cursor on the name of the inner class you want to make static, or select the class in the Structure view.
2. On the main menu, or on the context menu of the selection, choose Refactor | Make Static .
3. If the class references any outer class fields, the Make Class Static dialog suggests to pass the outer class as a parameter to the inner class constructor.
4. Preview results in the Find tool window and apply changes.

Make Method Static

The Make Method Static refactoring converts an instance method to a static one and automatically corrects all calls, implementations and overridings of the method.

> **Tip** This refactoring is also available from UML Class diagram .

- Examples
- Performing the Refactoring
- Make Static refactoring for a method in a call hierarchy

Examples

BeforeAfter

```
class ConnectionPool {
                    public int i;
                    public int j;
                    public void getConnection()
                    {
                    ...
                    }
                    }
```

```
class ConnectionPool {
                    public int i;
                    public int j;
                    public static void getConnection(ConnectionPool connectionPool) {
                    ...
                    }
                    }
```

```
class ConnectionPool {
                    public int i;
                    public int j;
                    public void getConnection()
                    {
                    ...
                    }
                    }
```

```
class ConnectionPool {
                    public int i;
                    public int j;
                    public static void getConnection(int i, int
                    j) {
                    ...
                    }
                    }
```

Performing the Refactoring

1. Select the method to be refactored in the Structure view, or right-click the method name in the editor. On the main menu, or on the context menu of the selection, choose Refactor | Make Static. The Make Method Static dialog box opens.
2. If the method references any of the containing class fields, do one of the following:
   - To pass the whole referenced object as a parameter to the method, select the Add object as a parameter with name check box and enter the name for the parameter.
   - To pass the referenced fields/variables as parameters to the method, select the Add parameters for fields check box and select the appropriate fields in the list. You can also change the order of the parameters using the Move Up and Move Down buttons.
3. If the method does not contain any references to fields or instance variables, you should only specify whether you want to replace instance qualifiers with class references.
4. To preview the results, click Preview and examine the reuslt of the refactoring in the Find tool window . Apply the changes, if no issues arizs.

Make Static refactoring for a method in a call hierarchy

In call hierarchies, if the method callers don't contain any other references to instance members, IntelliJ IDEA suggests that you make those callers static too.

Example

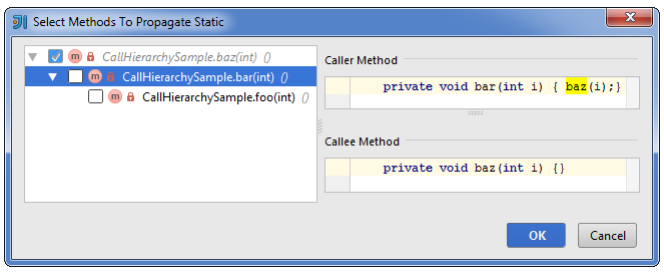
BeforeAfter

```
class CallHierarchySample {
                    private void foo(int
                    i) { bar(i);}
                    private void bar(int
                    i) { baz(i);}
                    private void baz(int
                    i) { }
                    }
```

```
class CallHierarchySample {
                    private static void foo(int i) { bar(i);}
                    private static void bar(int i) { baz(i);}
                    private static void baz(int i) { }
                    }
```

In this example, the refactoring was performed on `baz(int i)` . All the caller methods were selected for making static too.

When performing the refactoring, the Select Methods to Propagate Static dialog is shown. This dialog lets you select the caller methods to be made static.



Migrate

The Migrate refactoring allows you to easily switch between the old packages and classes used in your project and the new ones. IntelliJ IDEA comes with the default migration map. You can use it to provide Swing 1.1 support in a project that includes Swing 1.0.3 packages and classes.

Tip This refactoring is also available from UML Class diagram .

To perform migrate refactoring, follow these general steps

1. On the main menu, choose Refactor | Migrate. The Package and Class Migration dialog box opens.
2. Select the desired migration map from the drop-down list, or click the New button to create a new one. Edit Migration Map dialog appears.
3. Specify the name of the map, and optional map description.
4. Create a new migration description. To do that, click Add.
5. In the Edit Class/Package Migration Description dialog box that opens, specify the following options:
   - Select whether you want to migrate packages or classes.
   - Specify the old and the new names of the package or class.

   Click OK to apply changes and close the dialog.
6. Create or edit as many migration descriptions as required, and click OK to apply changes and close the Edit Migration Map dialog.
7. Click Run to start migration with the selected map.

Move Refactorings

On this page:

- Move Refactorings: Basics
- Performing Move Refactoring
- Moving a Package

Move Refactorings: Basics

Move refactorings allow you to move packages and classes between the source roots of a project, class members to other classes and inner classes to upper hierarchy levels. The move refactorings automatically correct all references to the moved packages, classes and members in the source code.

The following move refactorings are available:

- Move Package moves a package and its contents to another package or directory under the specified source root. When you move a package, you can choose between the following refactorings:
  - Move package to another package.
  - Move directory to another source root.
  - Move directory to another directory.

> **Note** The Move Package refactoring is significantly different from the other move refactorings. For corresponding instructions, see Moving a package . For all the other symbols, refer to Performing a Move refactoring .

- Move Class refactoring enables you to:
  - Move a class to a package under the specified source root.
  - Make a class an inner class.
- Move Static Members moves a static field, method or inner class to another class.
- Move Inner to Upper Level:
  - In Java, this refactoring moves an inner class to a higher level.
  - In ActionScript, this refactoring moves out-of-package classes, functions, variables, constants and namespaces into a package. (In this case the word inner is used to refer to entities (classes, functions, etc.) that are declared outside of packages. The upper level means a package.)
- Move Instance Method refactoring moves an instance method to another class.
- Move File refactoring moves a file to another directory.

> **Tip** This refactoring is also available from UML Class diagram .

Performing Move Refactoring

To perform a Move refactoring, follow these general steps:

1. Select the symbol to be moved and do one of the following:
   - On the Refactor menu, or on the context menu, choose Move.
   - Press F6 .
   - In the Project tool window , drag the symbol to the new destination.

   The dialog that opens depends on the type of the selected symbol.
2. Specify the move options according to the type of the item to be moved. See option descriptions in the Move dialog box reference.
3. Preview and apply the changes .

Moving a Package

To move a package, follow these steps:

1. Select the package in the Project tool window, and press F6 , or just drag the selected package.
2. In the Select Refactoring dialog box, click one of the options to specify which refactoring should be performed.
   - To move the whole package to another package select the first option, click OK, then specify the move options in the Move dialog box.
   - To move the directory to another source root, select the second option, click OK, and specify the destination source root.
   - To move the directory to another directory, select the third option, click OK, and specify the destination directory.

Pull Members Up

– Basics
– Example
– Pulling members up

Basics

The Pull Members Up refactoring allows you to move class members to a superclass or an interface, or interface to a superinterface .

Example

Before After

```
// File Class.java
public class Class extends
SuperClass {
public void publicMethod()
{
...
}
public void hiddenMethod()
{
...
}
}

// File SuperClass.java
public abstract class SuperClass
{
public abstract void publicMethod();
}
```

```
// File Class.java
public class Class extends
SuperClass {
public void publicMethod()
{
...
}
}

// File SuperClass.java
public abstract class SuperClass
{
public abstract void publicMethod();
public void hiddenMethod()
{
...
}
}
```

Pulling members up

To pull members up

1. Select the class to be moved to a superclass or interface.
2. On the main menu or on the context menu, choose Refactor | Pull Members Up. The Pull Members Up dialog box appears.
3. Select the destination object (interface or superclass).
4. In the Members section, select the members you want to move.
5. To move a method as abstract, select the check box in the column Make abstract next to the method in question.
6. In the JavaDoc section, select the action to be applied on JavaDoc.
7. Click Refactor to pull the selected members to their destination.

Push Members Down

The Push Members Down refactoring helps clean up the class hierarchy by moving class members to a subclass or a subinterface. The members are then relocated into the direct subclasses/interfaces only.

Example

Before **After**

```
// File Class.java
public class Class extends
SuperClass {
public void publicMethod()
{
...
}
}

// File SuperClass.java
public abstract class SuperClass
{
public abstract void publicMethod();
public void hiddenMethod()
{
...
}
}
```

```
// File Class.java
public class Class extends
SuperClass {
public void publicMethod()
{
...
}
public void hiddenMethod()
{
...
}
}

// File SuperClass.java
public abstract class SuperClass
{
public abstract void publicMethod();
}
```

To push members down

1. In the editor open a class, whose members you need to push down.
2. On the main menu or on the context menu, choose Refactor | Push Members Down. Push Members Down dialog box displays the list of members to be pushed down.
3. In the Members to be pushed down area, select the members you want to move. Note that the member at caret is already selected.
   If pushing a member might cause problems, you will be notified with red highlighting. It means that, if the situation is unattended, an error will emerge after refactoring. IntelliJ IDEA prompts you with a Problems Detected dialog, where you can opt to ignore or fix the problem.
4. Select the Keep abstract check box to:
   - Convert the original method to abstract, and move the original method body to the new method in subclass as an abstract method implementation, if the original method is non-abstract.
   - Create the new abstract method in a subclass and the same abstract method in subclass/subinterface (with possible errors if the subclass is not abstract), if the pushed down methods are already abstract.
5. When you push down abstract methods that have JavaDoc comments, specify to how treat them in the JavaDoc section.
6. Preview and apply changes.

Remove Middleman

The Remove Middleman refactoring allows you to replace all calls to delegating methods in a class with the equivalent calls directly to the field delegated to. Additionally, you can automatically remove the classes delegating methods, which will now be unused.

This refactoring is useful if you have a class that simply forwards many of its method calls to objects of other classes, and you wish to simplify your design.

Tip This refactoring is also available from UML Class diagram .

Example

Before After

```
public class Foo
                    {
                    Bar bar;
                    public Foo getImpValue() {
                    return bar.getImpValue();
                    }
                    }

                    public class Bar {
                    private Foo impValue1;
                    public Bar(Foo impValue) {
                    impValue1 = impValue;
                    }
                    public Foo getImpValue() {
                    return impValue1;
                    }
                    }

                    public class Client {
                    Foo a;
                    Foo impValue = a.getImpValue();
                    }
```

```
public class Foo
                    {
                    Bar bar;
                    public Bar getbar() {
                    return bar;
                    }
                    }

                    public class Bar {
                    private Foo impValue1;
                    public Bar(Foo impValue) {
                    impValue1 = impValue;
                    }
                    public Foo getImpValue(){
                    return impValue1;
                    }
                    }

                    public class Client {
                    Foo a;
                    Foo impValue = a.getbar().getImpValue();
                    }
```

To remove a middleman

1. Open the class in question in the editor, and position the caret at the name of the delegating field. Alternatively, select the delegating field in the Structure view of the desired class.

2. Choose Refactor | Remove Middleman on the main menu or on the context menu of the selection.

3. In the Remove Middleman dialog box:
   - Select whether you wish any methods which simply forward to calls to this field to be delegated. These methods will be unused by code in the current project, but may still be needed by code outside the project. You may wish to retain these delegating methods for backward compatibility.
   - Click Preview to make IntelliJ IDEA search for usages of the selected field, and display the refactoring preview results in the Find tool window . In the preview, you can include usages into refactoring or skip them. Click Do Refactor to apply refactoring to the selected usages.

     If you do not want to view usages, click Refactor. In this case, the usages will be changed immediately.

   The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Rename Refactorings

Rename refactorings allow you to rename symbols with all the references to them in the code corrected automatically.

The following rename refactorings are available in IntelliJ IDEA:

- Rename Package. The following usages are renamed:
  - Package statements
  - Import statements
  - Qualified names of classes
- Rename Class. The following usages are renamed:
  - Import statements
  - Qualified names of classes
  - Variables with the selected class type
  - Class inheritors
- Rename Method. The following usages are renamed:
  - All calls of the method.
  - All overridden/implemented methods in subclasses.
- Rename Field.
- Rename Function.
- Rename Variable.
- Rename Parameter. The following usages are renamed:
  - All usages of the parameter.
  - The corresponding `param` tag in documentation comment.
- Rename CSS color value.
- Rename File.
- Rename Directory.
- Rename views and references to views in the Grails applications.

Example

Before **After**

```
public class MyClass
                    {
                    // some code here
                    }
                    ...
                    public void myMethod() {
                    MyClass myClass = new MyClass();
                    }
```

```
public class  YourClass
                    {
                    // some code here
                    }
                    ...
                    public void myMethod() {
                    YourClass yourClass = new YourClass();
                    }
```

To rename a symbol, follow these general steps

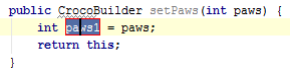
1. Select the item to be renamed.
   - To select a file, click the desired file in the Project tool window .
   - To select a symbol in the editor, place the caret at the name of the symbol to be renamed.
   - To select a symbol in the Project tool window , make sure that the members are shown, and then click the desired symbol.
   - To select a symbol in the Structure view , click the desired symbol in the Structure tool window.
2. Choose Refactor | Rename on the main menu or on the context menu, or press `Shift+F6` .
3. The subsequent behavior depends on the check box Enable in-place mode (Settings | Editor).
   - If this check box is selected, the suggested name appears right below the symbol in question. You can either accept suggestion, or type a new name. However, if you press `Shift+F6` once more, IntelliJ IDEA will display the Rename dialog box with more options.
   - If this check box is not selected, the Rename dialog box opens immediately.

   The set of controls and their names depend on the type of the symbol to be renamed.
4. If you want IntelliJ IDEA to find and rename objects related to the renamed class, whose names contain the changed string, check one or more of the following options:
   - Rename variables to rename the variables of that class type.
   - Rename inheritors to rename class inheritors.
   - Rename bound forms to rename the GUI forms bound to the class.

   If you chose to rename any of the objects bound to the renamed class, IntelliJ IDEA searches for appropriate items and displays them in a sequence of dialogs, sorted by type. In each dialog you may select the items you want to change.
5. Preview and apply changes .

Important notes

- Local variables are renamed in-place:



To be able to use the Rename dialog when renaming local variables, you should disable in-place refactoring in the editor settings .

Replace Constructor with Builder

The Replace Constructor with Builder refactoring helps hide a constructor, replacing its usages with the references to a newly generated builder class, or to an existing builder class.

Example

Before **After**

To replace a constructor with a builder class

1. Place the caret at the constructor invocation to be replaced.
2. Open Replace Constructor with Builder dialog box by choosing Refactor | Replace Constructor with Builder on the main menu, or on the context menu of the selection.

   The dialog box shows the list of constructor parameters to be passed to the builder, and the suggested name of a new builder.
3. In the Parameters to pass to the builder list, optionally change the suggested setter names, and specify whether you want to initialize the generated fields in the builder.

   If you specify an initial value that matches the parameter value in the constructor invocation, you can skip setter method for such parameter by selecting the Optional Setter check box.
4. Specify which builder are you going to use. If you select the Create new radio button, IntelliJ IDEA will generate a new builder, with the specified name, located in the specified package.

   If you select the Use existing radio button, find the desired builder class in your project.
5. To review the intended changes and make final corrections prior to the refactoring, click Preview. To apply the changes immediately, click Refactor.

Replace Constructor with Factory Method

The Replace Constructor With Factory Method refactoring allows you to hide a constructor and replace it with a static method which returns a new instance of a class.

Example

Before **After**

```
// File Class.java
public class Class {
public Class(String s) {
...
}
}

// File AnotherClass.java
public class AnotherClass {
public void method() {
Class aClass = new Class("string");
}
}
```

```
// File Class.java
public class Class {
private Class(String s) {
...
}
public static createClass(String s) {
return new Class(s);
}
}

// File AnotherClass.java
public class AnotherClass {
public void method() {
Class aClass = Class.createClass("string");
}
}
```

To replace a constructor with a factory method

1. Select the class constructor.
2. Open Replace Constructor With Factory Method dialog by choosing Refactor | Replace Constructor With Factory Method on the main menu or on tne context menu of the selection.
3. In the Factory method name field, specify the name for the factory method.
4. In the In (fully qualified name) field, specify the class, where the method should be created.
5. To review the intended changes and make final corrections prior to the refactoring , click Preview. To apply the changes immediately, click Refactor.

Replace Inheritance with Delegation

The Replace Inheritance With Delegation refactoring allows removing a class from inheritance hierarchy, while preserving the functionality of the parent. IntelliJ IDEA creates a private inner class, that inherits the former superclass or interface. Selected methods of the parent are invoked via the new inner class.

Example

Before **After**

```
// File Class.java
public class Class extends
SuperClass {
public int varInt;
public void openMethod() {
...
}
}

// File SuperClass.java
public abstract class SuperClass
{
public static final int CONSTANT=0;
public abstract void openMethod();
public void secretMethod()
{
...
}
}
```

```
// File Class.java
public class Class {
public int varInt;
private final MySuperClass superClass
= new MySuperClass();
public SuperClass getSuperClass()
{
return superClass;
}
public void openMethod() {
superClass.openMethod();
}
private class MySuperClass extends
SuperClass {
public void openMethod() {
...
}
}
}

// File SuperClass.java UNCHANGED
public abstract class SuperClass
{
public static final int CONSTANT=0;
public abstract void openMethod();
public void secretMethod()
{
...
}
}
```

To replace inheritance with delegation, follow these steps

1. Select the class to be refactored in the Project tool window , or open this class for editing and place the caret somewhere in the source code of the class.
2. On the main menu, or on the context menu of the selection, choose Refactor | Replace Inheritance With Delegation.
   The Replace Inheritance With Delegation dialog box opens.
3. In the Replace with delegation inheritance from field, select the parent object, inheritance to which will be replaced.
4. Specify the name for the field of the new inner class.
5. In the Inner class name field, specify the name for the inner class definition.
6. In the Delegate members area, select the members of the parent class, that will be delegated through the inner class.
7. To create a getter for the inner class, select the Generate getter for delegated component check box.
8. To review the intended changes and make final corrections prior to the refactoring, click Preview . To apply the changes immediately, click Refactor.

Find and Replace Code Duplicates

The Find and Replace Code Duplicates action allows you to find code repetitions similar to the selected method or constant field, and replace them with calls to the original method or constant.

Example

Before **After**

```
public void method() {
                    int a = 1;
                    int b = 2;
                    int c = a+b;
                    int d = b+c;
                    }
                    ...
                    private int add(int
                    a, int b)
                    {
                    return a+b;
                    }
```

```
public void method() {
                    int a = 1;
                    int b = 2;
                    int c = add(a,b);
                    int d = add(b,c);
                    }
                    ...
                    private int add(int
                    a, int b)
                    {
                    return a+b;
                    }
```

To find and replace code duplicates

1. Position the cursor within the method or a constant field whose duplicates you want to search for.
2. Select Refactor | Find and Replace Code Duplicates from the main or context menu.
3. In the dialog that opens, select the scope where IntelliJ IDEA shall look for code duplicates.
4. For each found code duplicate, IntelliJ IDEA will prompt you to confirm the replacement.

Replace Temp With Query

This refactoring allows you to extract the variable's initializer expression into a method, and replace all references to the variable with the calls to the extracted method. The declaration of the variable will be removed and the query method can be used in other methods.

Instead of `int size = getActualSize()` and using size throughout the code, we just operate with `getActualSize()` method. Though the resulting code has more invocations, it is much cleaner and helps identify precisely where the bottlenecks in the code can appear.

Example

Before **After**

```
public void method() {
                    String str ="str";
                    String aString = returnString().concat(str);
                    System.out.println(aString);
                    }
```

```
public void method() {
                    String str ="str";
                    System.out.println(aString(str));
                    }
                    private String aString(String str) {
                    return returnString().concat(str);
                    }
```

To replace temp with query

1. In the editor, position the caret at the name of the local variable you want to be refactored.
2. On the main menu, or on the context menu of the selection, choose Refactor | Replace Temp with Query . Replace Temp with Query dialog box appears. When selecting a local variable, make sure that its value is not modified later in the code. Otherwise the error message appears.
3. Specify the name for the extracted method.
4. To declare the method static, select the Declare static check box. This option is enabled when the initial expression is static.
5. In the Parameters section, select the parameters to be used in the extracted method. The parameters are all checked by default. If unchecked, the appropriate value will be used as a local variable in the extracted method.
6. Check the result in the Signature Preview pane and click OK to create the method.

Safe Delete

The Safe Delete refactoring lets you safely remove files and symbols from the source code.

To make sure that deletion is safe, IntelliJ IDEA looks for the usages of the symbol being deleted. If such usages are found, you can explore them and make the necessary corrections in your code before the symbol is actually deleted.

Tip This refactoring is also available from UML Class diagram .

- Performing the refactoring
- Safe Delete Parameter refactoring for a call hierarchy
- Safe Delete refactoring for a method in a call hierarchy

Performing the refactoring

1. Select the item to be deleted.
2. Do one of the following:
   - Press `Alt+Delete`.
   - Select Refactor | Safe Delete from the main or the context menu.
   - Select Refactor | Refactor This from the main menu (`Ctrl+Shift+Alt+T`), and select Safe Delete.
3. In the Safe Delete dialog , select the necessary options and click OK.
4. If the refactoring is potentially unsafe, the Usages Detected dialog opens.
   - View Usages. Click this button to see where in your code the item you are about to delete is used. As a result, the Find tool window opens.
     Analyze you code and make the necessary corrections. Then click Do Refactor. (If you want to rerun the refactoring from its start, click Rerun Safe Delete. IntelliJ IDEA will check if the refactoring is safe once more.)
   - Delete Anyway. Click this button to delete the item without looking at its usages.

Safe Delete Parameter refactoring for a call hierarchy

If a parameter is only passed through a call hierarchy and isn't used anywhere outside of that hierarchy, the Safe Delete refactoring lets you propagate the parameter deletion all along the hierarchy.

Example

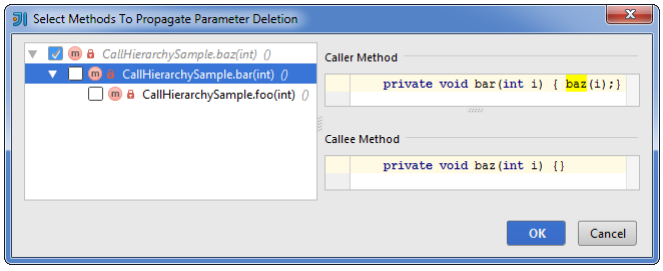
Before **After**

```
class CallHierarchySample {
                    private void foo(int
                    i) { bar(i);}
                    private void bar(int
                    i) { baz(i);}
                    private void baz(int
                    i) { }
                    }
```

```
class CallHierarchySample {
                    private void foo() {
                    bar();}
                    private void bar() {
                    baz();}
                    private void baz() {
                    }
                    }
```

In this example, the refactoring was performed on the `i` parameter within `baz(int i)`. This change was propagated to all the caller methods.

When performing the refactoring, the Select Methods to Propagate Parameter Deletion dialog is shown. This dialog lets you select the caller methods in which the parameter should be deleted.



Safe Delete refactoring for a method in a call hierarchy

When you perform the Safe Delete refactoring for a method, IntelliJ IDEA analyzes the corresponding call hierarchy, finds the methods that may become unused and suggests that you delete those methods too.

Example

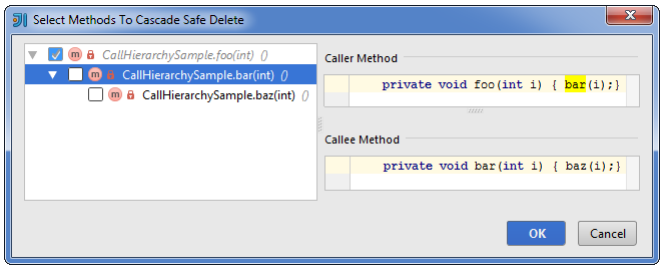
Before **After**

```
class CallHierarchySample {
                    private void foo(int
                    i) { bar(i);}
                    private void bar(int
                    i) { baz(i);}
                    private void baz(int
                    i) { }
                    }
```

```
class CallHierarchySample {



                    }
```

In this example, the refactoring was performed on the `foo(int i)` method. All the methods lower down in the call hierarchy were selected for deletion.

When performing the refactoring, the Select Methods to Cascade Safe Delete dialog is shown. This dialog lets you select the methods to be deleted.



Type Migration

The Type Migration refactoring allows you to automatically change a member type (e.g. from integer to string), and data flow dependent type entries, like method return types, local variables, parameters etc. across the entire project. It also lets automatically convert variable or method return type between arrays and collections. If any conflicts are found IntelliJ IDEA warns you about them.

Tip This refactoring is also available from UML Class diagram .

Example

`f: int -> String`

Before **After**

```
int f;
                    void bar(int i) {}
                    void foo() {
                    bar(f);
                    }
```

```
String f;
                    void bar(String i) {}
                    void foo() {
                    bar(f);
                    }
```

`I<String> -> I<Integer>`

Before **After**

```
interface I<T>
                    {
                    T foo(T t);
                    }

                    class A implements I<String> {
                    String myString;
                    public String foo(final
                    String s) {
                    if (s == null) {
                    return
                    myString;
                    }
                    return s;
                    }
                    }
```

```
interface I<T>
                    {
                    T foo(T t);
                    }

                    class A implements I<Integer> {
                    Integer myString;
                    public Integer foo(final
                    Integer s) {
                    if (s == null) {
                    return
                    myString;
                    }
                    return s;
                    }
                    }
```

`myResult: ArrayList<String> -> String[]`

Before **After**

```
public class ResultContainer
                    {
                    private ArrayList<String> myResult;
                    public String[] getResult() {
                    return myResult.toArray(new String[myResult.size()]);
                    }
                    }
```

```
public class ResultContainer
                    {
                    private String[] myResult;
                    public String[] getResult() {
                    return myResult;
                    }
                    }
```

To change type, follow these steps

1. In the editor, place the caret on the type to be refactored.
2. On main menu, choose Refactor | Type Migration, or press `Ctrl+Shift+F6`.
3. In the Type Migration dialog box, specify the new type and scope where to look for the usages.
4. Click Preview and review the items that will be affected. If needed, exclude usages from refactoring. To do that, right-click usage in the Type Migration Preview and select Exclude. When done, click Migrate.

Use Interface Where Possible

Use Interface Where Possible refactoring delegates execution of the specified methods, derived from a base class/interface, to an instance of an ancestor class or an inner class, implementing the same interface.

Example

Before After

```
// File Class.java
public class Class implements
Interface {
public void publicMethod()
{
...
}
public void hiddenMethod()
{
...
}
}
```

```
// File Class.java UNCHANGED
public class Class implements
Interface {
public void publicMethod()
{
...
}
public void hiddenMethod()
{
...
}
}
```

```
// File Interface.java
public interface Interface {
int CONSTANT=0;
void publicMethod();
}
```

```
// File Interface.java UNCHANGED
public interface Interface {
int CONSTANT=0;
void publicMethod();
}
```

```
// File AnotherClass.java
public class AnotherClass {
Class firstClass;
Class secondClass;
public void method() {
firstClass.publicMethod();
firstClass.hiddenMethod();
secondClass.publicMethod();
if (secondClass instanceof Class) {
...
}
...
}
}
```

```
// File AnotherClass.java MODIFIED
public class AnotherClass {
Class firstClass;
Interface secondInterface;
public void method() {
firstClass.publicMethod();
firstClass.hiddenMethod();
secondInterface.publicMethod();
if (secondInterface instanceof
Interface) {
...
}
...
}
}
```

To use interface where possible, follow these steps

1. Select a class whose methods should be delegated to its parent class or interface. To do that, place the caret on this class in the editor or in the Project tool window .
2. On the main menu or on the context menu of the selection, choose Refactor | Use Interface Where Possible .
3. In the Use Interface Where Possible dialog box, select the parent object that will replace the usages of the current class.
4. To replace the current class name in `instanceof` statements, check the option Use interface/superclass in instanceof.
   Note that if you use `instanceof` statements and leave this check box unselected, you may receive erroneous code, such as:

   ```
   if (secondInterface
                       instanceof
                       Class)
   ```

   This code will compile, but may produce undesired results.
5. To review the intended changes and make final corrections prior to the refactoring, click Preview . To continue without preview, click Refactor.
   The Rename variables dialog box appears. It lists the occurrences of the class that may be replaced by the superclass or the interface selected. Select the usages you want to replace, and (optionally) specify new names for each of them.
6. Click OK to continue. If you have previously clicked the Preview button, the Preview window will appear now.

Wrap Return Value

The Wrap Return Value refactoring allows you to select a method, and either create a wrapper class for its return values, or use an existing, compatible wrapper class. All returns from the method selected will be appropriately wrapped, and all calls to the method will have their returns unwrapped.

Wrapping a method's returns are useful, if your design changes in such a way that you want a method to return more information than originally planned. After wrapping, the wrapper class can be extended, allowing more data to be returned from the method. Also, it is common to wrap primitive return values, thus allowing interface and implementation to be decoupled as needed.

Tip This refactoring is also available from UML Class diagram .

Example

Before After

```
class Order {
                    String customer;
                    String getCustomer() {
                    return customer;
                    }
                    }
```

```
class Order {
                    String customer;
                    Customer getCustomer() {

                    }
                    }

                    class Customer {
                    String id;
                    Customer(String id)
                    {
                    this.id=id;
                    }
                    }
```

To wrap a return value, follow these steps

1. Open the desired class in the editor and place the caret at the method whose returns you wish to wrap.
2. Choose Refactor | Wrap Return Value on the main menu, or on the context menu of the selection. Alternatively, select the desired method in the Structure views, and trigger refactoring from there.
3. In the Wrap Return Value dialog box specify the name and package for the new wrapper class, or select an existing compatible wrapper class.
4. Preview and apply changes.

Tutorial. Replace conditional logic with strategy pattern

When you have a method with lots of conditional logic (i.e., if statements), you're asking for trouble. Conditional logic is notoriously difficult to manage, and may cause you to create an entire state machine inside a single method.

– Analyzing the sample application
– Creating and running test
– Extracting methods
– Using Extract Delegate refactoring
– Fine tuning
– Implementing the abstract class
– Happy end

Analyzing the sample application

Here's a short example. Let's say, there is a method that calculates insurance costs based on a person's income:

```
package ifs;

                    public class IfElseDemo {

                    public double calculateInsurance(double income) {
                    if (income <= 10000) {
                    return income*0.365;
                    } else if (income
                    <= 30000) {
                    return (income-10000)*0.2+35600;
                    } else if (income
                    <= 60000) {
                    return (income-30000)*0.1+76500;
                    } else {
                    return (income-60000)*0.02+105600;
                    }

                    }
                    }
```

Let's analyze this example. Here we see the four "income bands widths", separated into four calculation strategies. In general, they conform to the following formula:

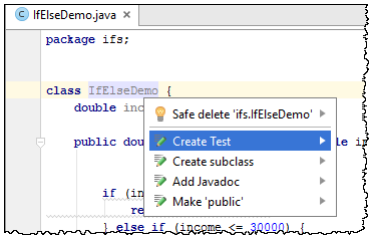
```
(income - adjustment)
                    * weight + constant
```

Our goal is to provide separate classes to calculate each strategy, and transform the original class to make it more transparent.

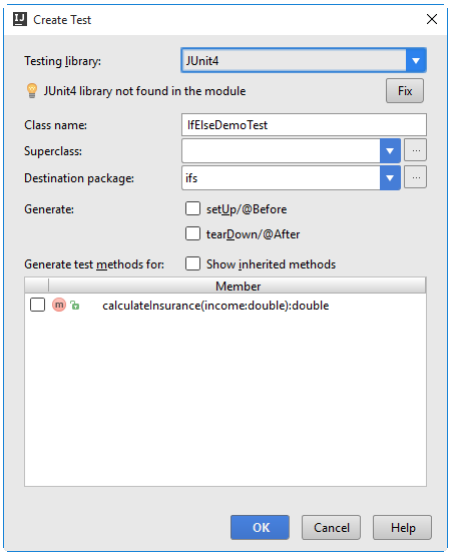
Creating and running test

First of all, let's make sure that the class works. To do that, create a test class, using the JUnit4 testing framework.

Place the caret at the class name, and then press `Alt+Enter` (or click). From the list of suggested intention actions, choose Create Test:



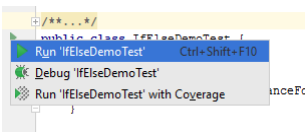
In the Create Test dialog, choose JUnit4 from the Testing library drop-down list, click Fix, if you do not have JUnit library, and then click OK:



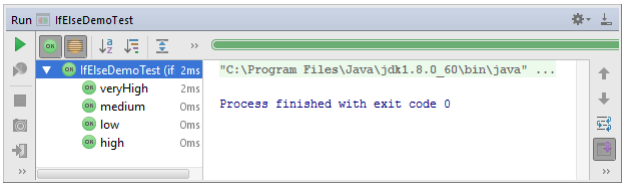
The stub test class is created. However, you have to provide some meaningful code, using the provided quick fix ( ) to create import statement:

```
import org.junit.Test;

                    import static org.junit.Assert.*;


                    public class IfElseDemoTest {
                    @Test
                    public void low() {
                    assertEquals(1825, insuranceFor(5000), 0.01);
                    }

                    @Test
                    public void medium() {
                    assertEquals(38600, insuranceFor(25000), 0.01);
                    }

                    @Test
                    public void high() {
                    assertEquals(78500, insuranceFor(50000), 0.01);
                    }

                    @Test
                    public void veryHigh() {
                    assertEquals(106400, insuranceFor(100_000), 0.01);
                    }

                    private double insuranceFor(double income) {
                    return new IfElseDemo().calculateInsurance(income);
                    }
                    }
```

Now let's run this test by clicking the run button in the left gutter, or by pressing `Ctrl+Shift+F10`:



All 4 tests pass:

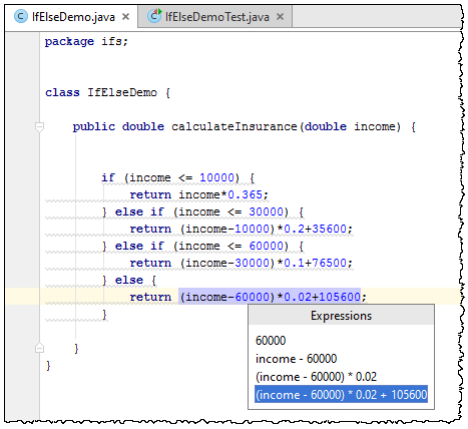


Extracting methods

Next, bring forward the original class, place the caret at the expression

```
(income-60000)*0.02+105600
```

and invoke Extract Method Dialog dialog (`Ctrl+Alt+M`):



You get the following code:

```
package ifs;


class IfElseDemo {

public double calculateInsurance(double income) {
if (income <= 10000) {
return income*0.365;
} else if (income
<= 30000) {
return (income-10000)*0.2+35600;
} else if (income
<= 60000) {
return (income-30000)*0.1+76500;
} else {
return calculateInsuranceVeryHigh(income);
}
}
public double calculateInsuranceVeryHigh(double income) {
return (income-60000)*0.02+105600;
}
}
```

Next, use the same Extract Method refactoring for for 60000, 0.02 and 105600 fragments, and create the methods `getAdjustment` , `getWeight` and `getConstant` :

```
package ifs;


                    class IfElseDemo {

                    public double calculateInsurance(double income) {
                    if (income <= 10000) {
                    return income*0.365;
                    } else if (income
                    <= 30000) {
                    return (income-10000)*0.2+35600;
                    } else if (income
                    <= 60000) {
                    return (income-30000)*0.1+76500;
                    } else {
                    return calculateInsuranceVeryHigh(income);
                    }
                    public double calculateInsuranceVeryHigh(double income) {
                    return (income- getAdjustment())* getWeight() + getConstant();
                    }
                    public int getConstant() {
                    return 105600;
                    }

                    public double getWeight() {
                    return 0.02;
                    }
                    public int getAdjustment()
                    {
                    return 60000;
                    }
                    }
```

Using Extract Delegate refactoring

Next, select all the methods created in the previous chapter, and invoke the Extract Delegate refactoring:

The newly created class has the name `InsuranceStrategyVeryHigh`.

Then delete all the selected methods from the class `IfElseDemo`. Thus you get the following two classes:

```
package ifs;


                    class IfElseDemo {

                    private final InsuranceStrategyVeryHigh insuranceStrategyVeryHigh
                    = new InsuranceStrategyVeryHigh();

                    public double calculateInsurance(double income) {


                    if (income <= 10000) {
                    return income*0.365;
                    } else if (income
                    <= 30000) {
                    return (income-10000)*0.2+35600;
                    } else if (income
                    <= 60000) {
                    return (income-30000)*0.1+76500;
                    } else {
                    return insuranceStrategyVeryHigh.calculateInsuranceVeryHigh(income);
                    }

                    }
```

and

```
package ifs;

                    public class InsuranceStrategyVeryHigh {
                    public InsuranceStrategyVeryHigh() {
                    }

                    public double calculateInsuranceVeryHigh(double income) {
                    return (income - getAdjustment()) * getWeight() +
                    getConstant();
                    }

                    public int getConstant() {
                    return 105600;
                    }

                    public double getWeight() {
                    return 0.02;
                    }

                    public int getAdjustment()
                    {
                    return 60000;
                    }
                    }
```

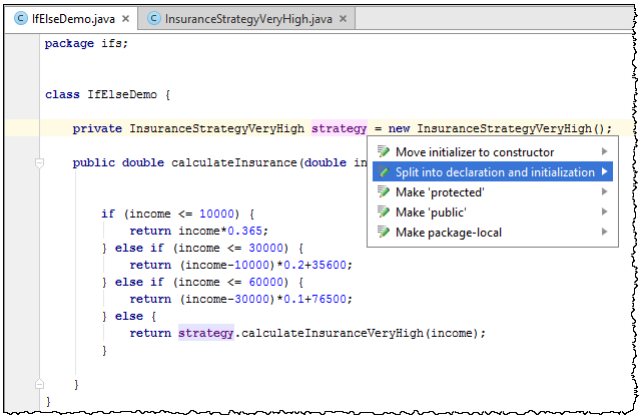
Fine tuning

This code requires some modifications. First, let's change the class `IfElseDemo`:

– Rename (`Shift+F6`) the field `insuranceStrategyVeryHigh` to `strategy`

– Make this field not final.

– Using the intention action, split it into declaration and initialization:

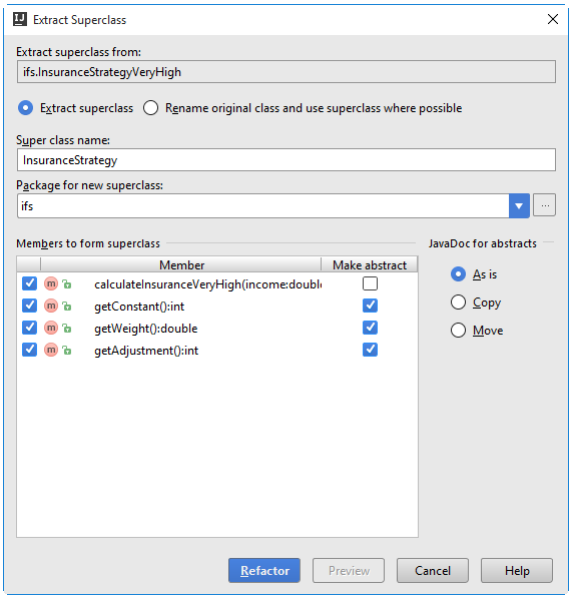


– Move the initialization down to the corresponding `if-else` branch.

Get the following code:

```
package ifs;

                    class IfElseDemo {

                    private InsuranceStrategyVeryHigh strategy;

                    public double calculateInsurance(double income) {


                    if (income <= 10000) {
                    return income*0.365;
                    } else if (income
                    <= 30000) {
                    return (income-10000)*0.2+35600;
                    } else if (income
                    <= 60000) {
                    return (income-30000)*0.1+76500;
                    } else {
                    strategy = new InsuranceStrategyVeryHigh();
                    return strategy.calculateInsuranceVeryHigh(income);
                    }

                    }

                    }
```

Next, let's modify the class `InsuranceStrategyVeryHigh` - invoke the Extract Superclass refactoring for it.



Mind the settings in the Extract Superclass Dialog dialog:

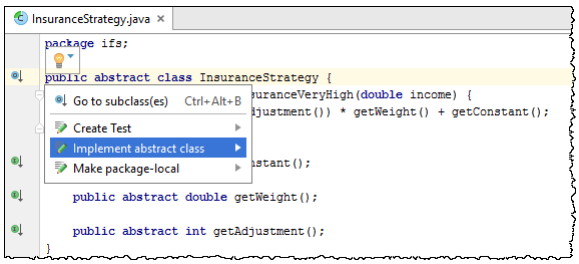
- The name of the superclass to be generated is `InsuranceStrategy`.
- All the methods of the class `InsuranceStrategyVeryHigh` are checked - it means that they will be included in the superclass.
- The method `calculateInsuranceStrategyVeryHigh` remains non-abstract; all the other methods are made abstract by selecting the Make Abstract check boxes.

Agree to replace the usage of `InsuranceStrategyVeryHigh` class (in `IfElseDemo` class) with the superclass, and get the following `InsuranceStrategy` class:

```
package ifs;

                    public abstract class InsuranceStrategy
                    {
                    public double calculateInsuranceVeryHigh(double income) {
                    return (income - getAdjustment()) * getWeight() +
                    getConstant();
                    }

                    public abstract int getConstant();

                    public abstract double getWeight();

                    public abstract int getAdjustment();
                    }
```

Implementing the abstract class

Next, for this abstract class, use Implement Abstract Class intention to create implementations for all strategies:



Name the new implementation classes `InsuranceStrategyLow`, `InsuranceStrategyMedium` and `InsuranceStrategyHigh`.

For all new implementations provide correct `return` statements for the methods `getAdjustment()`, `getWeight()` and `getConstant()`.

Thus all implementation classes should look similar to the class `InsuranceStrategyVeryHigh`, but with strategy-specific adjustment, weight and constant. For example:

```
package ifs;

                    public class InsuranceStrategyMedium extends
                    InsuranceStrategy {
                    @Override
                    public int getConstant() {
                    return 35600;
                    }

                    @Override
                    public double getWeight() {
                    return 0.2;
                    }

                    @Override
                    public int getAdjustment()
                    {
                    return 10000;
                    }
                    }
```

Note that in all newly created implementation classes the class names are grey - they are not used so far.

Next, bring forward the class `IfElseDemo`, and modify all the branch bodies so that they initialize the `strategy` field, like in the last branch:

```
package ifs;

                    class IfElseDemo {

                    private InsuranceStrategy strategy;

                    public double calculateInsurance(double income) {


                    if (income <= 10000) {
                    strategy = new InsuranceStrategyLow();
                    return strategy.calculateInsuranceVeryHigh(income);
                    } else if (income
                    <= 30000) {
                    strategy = new InsuranceStrategyMedium();
                    return strategy.calculateInsuranceVeryHigh(income);
                    } else if (income
                    <= 60000) {
                    strategy = new InsuranceStrategyHigh();
                    return strategy.calculateInsuranceVeryHigh(income);
                    } else {
                    strategy = new InsuranceStrategyVeryHigh();
                    return strategy.calculateInsuranceVeryHigh(income);
                    }

                    }

                    }
```

Finally, rename the method `calculateInsuranceVeryHigh`: bring forward the class `InsuranceStrategy`, place the caret at the method name and press `Shift+F6`.

The new name should be `calculate`.

Happy end

And finally enjoy the code:

```
package ifs;

                    class IfElseDemo {

                    private InsuranceStrategy strategy;

                    public double calculateInsurance(double income) {
                    if (income <= 10000) {
                    strategy = new InsuranceStrategyLow();
                    return strategy.calculate(income);
                    } else if (income
                    <= 30000) {
                    strategy = new InsuranceStrategyMedium();
                    return strategy.calculate(income);
                    } else if (income
                    <= 60000) {
                    strategy = new InsuranceStrategyHigh();
                    return strategy.calculate(income);
                    } else {
                    strategy = new InsuranceStrategyVeryHigh();
                    return strategy.calculate(income);
                    }
                    }
                    }
```

Next, let's run the test class again. All tests should pass – we have refactored the code, but it still produces the same results.

Using Local History

This section describes how to use Local History , which is your personal real-time version control system. Local History is independent of external version control systems and works with the directories of your project even when they are not under any VCS control.

Local history is cleared when you install a new version of IntelliJ IDEA or when you invalidate caches . Therefore, check in the changes to your version control system before performing these operations.

This section describes how to:

- View local history of a file or folder
- View local history of a class, method, field or selection
- View local history of a selection
- View recent changes
- Restore files from local history
- Mark local versions with labels

Putting Labels

On this page:

– Basics

– Adding a label to a local version

Basics

Before embarking on a risky change to your source code, it is a good idea to mark the stable version with some meaningful label. This will help you quickly roll back to a safe version.

Labels apply to a whole project.

Adding a label to a local version

To add a label to a local version

1. Select a file or folder in the Project tool window, or open a file in the editor.
2. Do one of the following:
   – On the main VCS menu, or on the context menu of the selection, choose Local History | Put Label.
   – Press `Alt+Back Quote` and choose Put Label command from the VCS Operations quick list.
3. In the Put Label dialog box, type the label name.

Restoring a File from Local History

Rolling back changes from the local history works same way as in the regular version control.

To roll back changes in the local history

1. Open the Local History view .
2. Select the version you want to roll back to.
3. On the context menu of the selection, choose Revert.

Viewing Local History of Source Code

On this page:

- Basics
- Viewing local history of a class
- Viewing local history of a source code block

Basics

Besides file history , you can track local changes for a class , its elements , or selected block of source code . his history shows only those changes that affect the selected element or code fragment.

Viewing local history of a class

To view local history of a class

1. Select a class file in the project tool window, or right-click class name in the editor.
2. On the main VCS menu, or on the context menu of the selection, choose Local History | Show History for Class.

To view local history of a method or field

1. In the editor, place the caret at the name of the desired method or field.
2. On the main VCS menu, or on the context menu of the selection, choose Local History | Show History for Method (Field).

Viewing local history of a source code block

To view local history of a source code block

1. In the editor, select a fragment of source code.
2. Do one of the following:
   - On the main VCS menu, or on the context menu of the selection, choose Local History | Show History for Selection.
   - Press `Alt+Back Quote`, and choose the desired command from the VCS Operations quick list.

Viewing Local History of a File or Folder

On this page:

– Basics

– Viewing local history

Basics

Local History makes it possible to view changes made to a certain file or a whole directory. Each entry in the Local History dialog box is displayed with its time stamp, action and optional label. So doing, the local history for a file includes all changes that affect both the selected file and the whole project; local history for a folder shows changes to the source code tree in general. You can explore changes, selecting the respective row in the Local History dialog box.

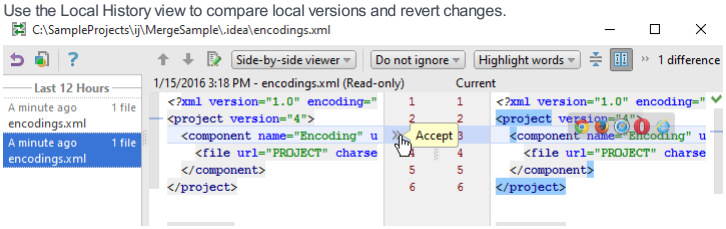
Viewing local history

To view local history

1. Select a folder or file in the Project tool window, or open a file in the editor.

2. Do one of the following:
   – On the main VCS menu, or on the context menu of the selection, choose Local History | Show History.
   – Press `Alt+Back Quote`, and choose the desired command from the VCS Operations quick list.
   – Use View Recent Changes that shows a summary of recent changes in a single pop-up list. Clicking an entry in this list shows the respective Local History.

3. Use the Local History view to compare local versions and revert changes.



Viewing Recent Changes

On this page:

– Basics

– Viewing recent changes

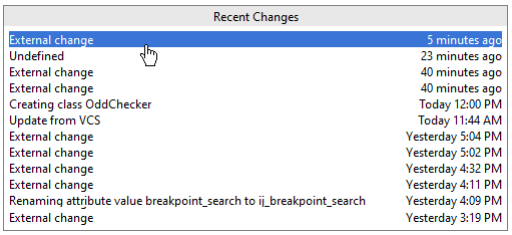
Basics

IntelliJ IDEA enables you to view the summary of the recent changes to a project. From the Recent Changes pop-up window, you can browse through the history of changes, navigate to a particular change, compare versions, and roll changes back.

Viewing recent changes

To view recent changes

1. On the main menu, choose View | Recent Changes, or press `Shift+Alt+C`.
2. In the Recent Changes pop-up window, select the change you are interested in:



3. In the dialog box that opens, review differences and discard changes, if necessary.

Version Control with IntelliJ IDEA

In this part you can find descriptions of procedures that are common to all supported version control systems, or specific to certain VCS integrations:

- Concepts of Version Control
- Managing Projects Under Version Control
- Accessing VCS Operations
- Enabling Version Control
- Configuring Version Control Options
- Common Version Control Procedures
- VCS-Specific Procedures

Concepts of Version Control

IntelliJ IDEA provides integration with several Version Control Systems (referred to as VCS in documentation). This includes both support of features specific for each VCS as well as unified interface and management for common VCS tasks.

In this part:

– Supported Version Control Systems
– Unified Version Control Functionality
– Directory-Based Versioning Model
– Changelist
– Local, Repository, and Incoming Changes
– Local History
– Patches
– Shelved Changes

Supported Version Control Systems

The list of supported integrations of version control systems is determined by the set of currently enabled plugins .

The basic principles of getting started with the supported version control systems (VCS) in IntelliJ IDEA are rather similar, though some commands and settings are specific and conform to the version control system conventions. In addition to the common information, you can find VCS-specific procedures in the following sections:

- Using Git Integration
- Using GitHub Integration
- Using Subversion Integration
- Using Mercurial Integration
- Using CVS Integration
- Using Perforce Integration
- Using Visual SourceSafe Integration
- Using ClearCase Integration
- Using TFS Integration

Unified Version Control Functionality

In addition to support for general and individual VCS commands, IntelliJ IDEA provides several unique features that simplify and speed up the work with any version control system.

- For the projects with VCS support enabled, the standard VCS actions (commit, update, revert, show differences and show history) are added to the main toolbar.
- Commit and update an entire project.
- Uniform interface for configuring common version control system settings.
- Changelists support for all integrated version control systems.
- Next, Previous, Rollback, Show Difference actions are available from the dedicated gutter bar in changed locations.
- View revision history for file/directory.
- Automatic checkout of all affected files when refactoring.
- Advanced Version Control tool window, with multiple dedicated tabs: History, Status, Update Info, etc.

Mind the difference in terminology in the different version control systems. For example, to denote the check-in functionality, Git uses the term *commit*, Subversion uses *submit*, etc.

Directory-Based Versioning Model

IntelliJ IDEA adopts a directory based model for version control association. A version control system is assigned to a project directory and/or additional directories that are part of or related to the project. Directories under version control are not required to be located under the project root. They can reside in any accessible location.

Changelist

A changelist is a set of changes in files that represents a logical change in source code. The changes specified in a changelist are not stored in the repository until committed (pushed).

Any changes made to the source files, are automatically included in the active changelist. Initially, the Default changelist is active, but you can make any other changelist active. The active changelist is displayed on top of the Version Control tool window, with the name being highlighted in bold font.

In addition to the Default changelist, you can create new changelists, delete existing ones (except for the Default changelist), and move files between changelists.

All modified, deleted, unversioned and other files are managed in the Version Control tool window . From this window you can:

- Commit (push) changelists.
- Create new changelists (if you want to keep an eye on certain files and changes).
- Remove existing changelists and set the default changelists.
- Rollback modified files in changelists.
- Add the unversioned files and directories to the version control.
- Move files between changelists.
- Show differences on selected files.
- Refresh the list of VCS changes.
- Jump to the source code from within a changelist.
- Shelve (stash) and unshelve (unstash) changes.

Local, Repository, and Incoming Changes

IntelliJ IDEA distinguishes among three categories of changes: Local, Repository, and Incoming.

Local changes
: The changes that you have introduced to your local working copy but have not yet checked in to the repository.

Repository changes
: The changes that you and other users have checked in to the repository.

Incoming changes
: The changes that have been checked in to the repository but that you have not yet checked out locally.

> **Warning!** Repository and incoming changes are not supported by Git and Mercurial integrations.

All these types of changes are displayed in the respective tabs of the Version Control tool window. Information about changes is stored in the history cache. Configure the size of this cache and frequency of refreshing information in the General VCS Settings .

Local History

Your source code constantly changes as you edit, test, or compile. Any version control system tracks the differences between the committed versions, but the local changes between commits pass unnoticed. Local History is your personal version control system that tracks changes to your source code on your computer and enables you to compare versions and roll changes back, if necessary. Local History is always at your disposal, no steps are required to enable it.

Local History is independent of external version control systems and works with the directories of your project even when they are not under any VCS control. It applies to any structural artifacts: a project, a directory or package, a file, a class, class members, tags, or selected fragment of text.

Unlike usual version control systems, Local History is intended for your personal use, it does not support shared access.

With Local History, IntelliJ IDEA automatically tracks changes you make to the source code, results of refactoring, and state of the source code based on a set of predefined events (testing, deployment, commit or update).

Local History revisions are marked with labels, which are similar to versions in traditional version control systems. Labels based on predefined events are added to the local revisions automatically; besides that, you can put your own labels to the project artifacts to mark your changes. Reverting or viewing differences are performed against these labels.

Local History has certain limitations:

- Tracking local changes is only possible for textual files. Binary files do not have Local History.
- For files larger than 1 Mbyte, Local History tracks only the very fact of changes, but does not preserve the respective contents.
- Local History does not support shared access.

Patches

IntelliJ IDEA helps you create and apply patches to the source code. A patch is a file in the standard text format that has the *.patch extension, and contains a list of differences between two sets of source files.

Patches only contain changes to textual files. Changes to binary files cannot be patched.

This concept is tightly related to the concept of Shelved Changes .

Shelved Changes

You can run into a situation when you are short of time to bring your source code to a certain required condition or you need to work on an urgent high priority task. In this case you might want to put some changes aside and continue working on a stable version.

With IntelliJ IDEA, you can use shelves for storing postponed changes temporarily. In due time, the desired changes can be taken back from the shelf (unshelved).

IntelliJ IDEA enables shelving both separate files and entire changelists. Accordingly, you can unshelve entire shelves or specific files.

Managing Projects under Version Control

This section contains information related to sharing IntelliJ IDEA project files with the other developers:

- Directory based project format
- Legacy project format
- Sharing run/debug configurations
- Sharing inspection profiles
- Project settings files to share

Directory based project format

The project settings are stored in the `.idea` directory. This format is used by all the recent IntelliJ IDEA versions by default. Here is what you need to share:

- All the files under `.idea` directory in the project root except `workspace.xml` and `tasks.xml`, storing user-specific settings.
- All the `.iml` module files that can be located in different module directories.

Legacy project format

The project settings are stored in the `.ipr/.iml/.iws` files.

Share the project `.ipr` file and all the `.iml` module files, don't share the `.iws` file as it stores user specific settings.

Sharing run/debug configurations

You might want to share run/debug configurations. To do that, just select the check box Share in the selected run/debug configuration dialog box.

The shared run/debug configurations are kept in separate xml files under `.idea\runConfigurations` folder, while the local run/debug configurations are kept in the `.idea\workspace.xml`.

Sharing inspection profiles

To share inspection profiles, make sure to select the check box Share profile on the Inspections page of the Settings dialog.

The shared inspection profiles are stored in separate xml files under `.idea\inspectionProfiles` folder, while the local profiles are kept in the `.idea\workspace.xml`.

Project settings files to share

The `config` directory has several subfolders that contain xml files with your personal settings. You can easily share your preferred keymaps, color schemes, etc. by copying these files into the corresponding folders on another IntelliJ IDEA installation. Prior to copying, make sure that IntelliJ IDEA is not running, because it can erase the newly transferred files before shutting down.

The following is the list of some of the subfolders under the `config` folder, and the settings contained therein.

Folder name	User Settings
`codestyles`	Contains code style schemes.
`colors`	Contains editor colors and fonts customization schemes.
`filetypes`	Contains user-defined file types.
`inspection`	Contains code inspection profiles.
`keymaps`	Contains IntelliJ IDEA keyboard shortcuts customizations.
`options`	Contains various options, for example, feature usage statistics and macros.
`templates`	Contains user-defined live templates.
`tools`	Contains configuration files for the user-defined external tools.
`shelf`	Contains shelved changes.

Warning! Be careful about sharing the following:

- Android artifacts that produce a signed build, as they contain keystore passwords.
- `dataSources.ids`, `datasources.xml` - these files can contain database passwords.

Also, consider not sharing the following:

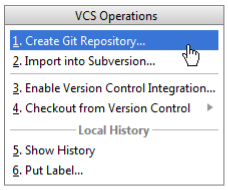
- The gradle.xml file.
- The user dictionaries folder to avoid conflicts if another developer has the same name.

Accessing VCS Operations

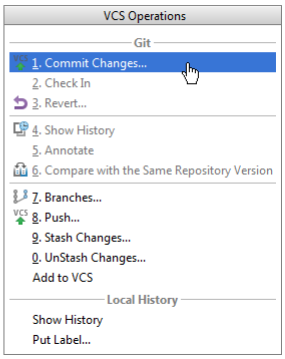
Use the VCS Operations Pop-up to quickly invoke most required commands.

Note that the composition of VCS commands available on the pop-up menu, depends on the specific VCS.

No VCS selected



VCS enabled



To quickly invoke a VCS command using VCS Operations Pop-up

1. Open VCS Operations pop-up, in one of the following ways:
   - On the main menu, choose VCS | VCS Operations Pop-up.
   - Press `Alt+Back Quote`.

2. Choose command from the VCS Operations list. To do that, perform one of the following actions:
   - Click the desired command in the list.
   - Use up and down arrow keys to select the desired command, and then press `Enter`.
   - Press the number key that corresponds to the desired command in the list.

Enabling Version Control

IntelliJ IDEA supports version control integration at two levels:

- At the IDE level, VCS integration is provided through a set of bundled plugins enabled by default.
- At the project level, VCS integration is enabled by associating project folders with one or several version control systems.

This section describes how to:

- Associate a project root with version control system.
- Associate a directory with version control system.
- Change the associated VCS for the project root or directory.

Associating a Project Root with a Version Control System

IntelliJ IDEA allows you to quickly enable your project's integration with a version control system, and associate it with the project root. For instructions on how to associate separate project directories with different version control systems, refer to Associating a Directory with a Specific Version Control System .

To assign a version control system to the project root

1. Choose VCS | Enable Version Control Integration on the main menu, or press `Alt+Back Quote`, and select Enable Version Control Integration....
2. In the Enable Version Control Integration dialog box that opens, select a version control system from the drop-down list that you want to associate with your project root .

Associating a Directory with a Specific Version Control System

IntelliJ IDEA supports a directory-based versioning model , which means that each project directory can be associated with a different version control system.

Associating a directory with a version control system

To associate a directory with a version control system, follow these steps:

1. Open version control settings (File | Settings | Version Control). This page shows a list of project directories and version control systems associated with them (if no directories have been added, the list only contains the project root).
2. Click the Add button + on the right.
3. In the Add VCS Directory Mapping dialog box that opens, select the Directory option. Type the path to the directory that you want to associate with a version control system, or click the Browse button ... and select the directory in the dialog that opens .
4. From the VCS drop-down list, select the version control system that will be used to control the files in this directory. Note that this list only contains the version control systems for which the corresponding Plugins are currently enabled (see Managing Plugins for details).
5. Optionally, click the Configure VCS button that allows you to specify the settings for the selected version control system. The same settings are also available under the Version Control settings node.
6. Click OK to save the mapping and return to the Version Control page.

Managing unregistered directories

For projects with Git or Mercurial integration enabled, IntelliJ IDEA scans project directories to check if there are Git/Mercurial repositories that are not controlled by the IDE. If such repositories are detected, IntelliJ IDEA displays a notification.

To add an unregistered root, click the Add roots link in the notification. Alternatively, open the Version Control settings page , select the unregistered root you want to add (they are marked grey) and follow the procedure Associating a Directory with a Specific Version Control System .

If you do not want to be notified about these roots again, click the Ignore link in the notification. Note that if new unregistered repositories are added to the project, IntelliJ IDEA will notify you about them.

To change the version control system associated with a directory

1. Open the Version Control settings page. This settings page displays a table of directories with associated version control systems.
2. In the table, locate the row that corresponds to the directory which you want to put under another version control system.
3. Click the VCS column. From the drop-down list that appears, select a new version control system.

   Tip Optionally, click the Configure VCS button. The Version Control Configurations dialog box opens where you can configure settings for the selected version control system (see the corresponding configuration settings in the Version Control settings page reference for details).

4. Click OK to save the new mapping and close the Version Control dialog box.

Configuring Version Control Options

Configuring version control options involves:

- Configuring General VCS Settings
- Configuring VCS-Specific Settings

Configuring General VCS Settings

General version control settings apply to all version control systems integrated with IntelliJ IDEA. General settings are specified on the Version Control page of the Settings dialog box, and include defining actions that require confirmation, background operations, ignored files, issue navigation, and depth of history.

To configure general version control settings, follow these general steps

1. Open the Settings dialog box and then click Version Control.
2. Specify which version control related actions should require confirmation .
3. Specify which operations should be performed in the background .
4. Create a list of files to be ignored by version control systems.
5. Configure history cache handling .
6. Define issue navigation rules to switch from check-in comments to corresponding issues in a bug tracking system.

Specifying Actions to Confirm

You can define that certain version control related activities should be performed only upon confirmation from your side. The activities for which confirmation is required are specified in the Confirmation settings page.

To specify which actions should require confirmation

1. Below the Version Control node in the settings, select the Confirmation page.

2. In the Files Creation/Deletion area, define how files created or deleted in IntelliJ IDEA should be added to or removed from a version control system. The available options are:
   - Show options before adding to version control
   - Add silently
   - Do not add

3. In the Display options dialog when these commands are invoked area, specify the commands, for which you want IntelliJ IDEA to display the Options dialog box. The available options are:
   - Check Out
   - Status
   - Get Latest Version
   - Update
   - Undo Check Out

4. Configure additional settings for working with changelists by selecting or clearing the corresponding check boxes:
   - Specify whether the read-only state of files should require explicit cancellation.
   - Specify whether meaningful comments on committing files to the repository should be required.
   - Specify whether uncommitted changes should be moved to another changelist.
   - Specify whether and how to create a changelist if the commit operation fails.

Specifying Actions to Run in the Background

You can enable background execution of certain version control related activities. These activities are specified in the Background settings page.

To specify the operations to run in the background

1. Below the Version Control node in the settings, select the Background page.
2. Enable background execution of the necessary actions by selecting the corresponding check boxes. Background execution can be set for the following actions:
   - Update
   - Commit
   - Checkout
   - Edit/Checkout
   - Add / Remove
   - Revert
   - History Cache Handling
   - Detecting "changed on server" conflicts

Configuring Ignored Files

On this page:

- Basics
- Defining a list of ignored files
- Patterns

Basics

Sometimes you may need to leave files of certain types unversioned. These can be VCS administration files, artifacts of utilities, backup copies, etc. You can create a global ignore list that will be stored in the workspace file and applied to all supported version control systems.

The files you want to ignore can be appointed explicitly by their names of through name patterns with wildcards. To ignore a directory, you need to specify the full path to it relative to the project root.

Use the Ignored Files settings page to list files that must be excluded from version control operations.

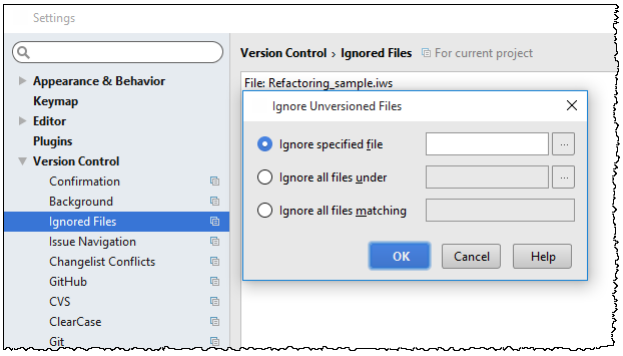
Tip If the version control system that you are using has its own ignore facilities, use the corresponding native command provided by the version control integration.

Note that once you've added a file to a version control system, ignoring it will have no effect. You need to remove it from your VCS first.

Defining a list of ignored files

1. Open the Ignored Files settings page by doing one of the following:
   - Under the Version Control node of the Settings dialog box, click Ignored Files.
   - In the Local Changes tab of the Version Control tool window, click the Configure Ignored Files toolbar button .

   The Ignored Files dialog box opens.



2. Click + (`Alt+Insert`) to create a new entry, or select an existing entry and click (`Enter`). The Ignore Unversioned Files dialog box opens.
3. Explicitly specify the files/directories to be ignored or define file name patterns. Do one of the following:
   - Choose the Ignore specified file option and specify the file name relative to the project root, for example, `my_folder/my_subfolder1/my_subfolder2/my_file` . Type the name manually or click the Browse button and select the desired file in the Select File to Ignore dialog box.
   - Choose the Ignore all files under option and specify the directory whose contents should be ignored. Type the directory name relative to the project root, for example, `my_folder/my_subfolder1/` , or click Browse button and select the desired folder in the Select Directory to Ignore dialog box.
     The rule is applied recursively to all the subdirectories of the specified directory. If a directory has several subdirectories and you want only one of them ignored, specify the required directory explicitly, for example, `my_folder/my_subfolder1/my_subfolder2/` .
   - Select the Ignore all files matching option and type the pattern that defines the names of files to be ignored. The rule is applied to all the directories under the project root. Using wildcards in combination with slashes (`/`) to restrict the scope to a certain directory is not supported.
     Patterns that define files to ignore, make use of two wildcards .
4. Create as many entries as you need and close the dialog box.

Tip You can also add files to the ignore list on-the-fly by using the Ignore command on the context menu of a newly added file under the Unversioned Files node in the Local Changes tab of the Version Control tool window .

Patterns

Two characters are used as the wildcards:

- `*` : this character matches anything.
- `?` : this character matches a single character.

For example, `*.iml` will ignore all the files with the extension `iml` ; `*.?ml` will ignore all the files with the extension ending with `ml` .

Configuring History Cache Handling

You can configure handling of the history cache in the Background settings page.

To configure history cache handling

1. Below the Version Control node in the settings, select the Background page.
2. Set the cache scope. Depending on the version control system you are using, do one of the following:
   - Specify the maximum number of changelists to be stored in the cache.
   - Specify the maximum number of days for a changelist to be stored in the cache.
3. Specify whether and how often (in minutes) you want your version control system to refresh the cache.

Configuring VCS-Specific Settings

Certain settings are applicable to the version control systems assigned to the whole project, or its directories . The others are related to the selected version control systems. Use the respective VCS pages under the VCSs node of the Settings dialog box to define VCS-specific settings.

To configure VCS-specific settings, follow these general steps

1. Open the Settings dialog box and then click Version Control.
2. Click a page that corresponds to the VCS to be configured.
3. Set up options as required. For detailed information, see VCS-specific pages of the Version Control settings.

Common Version Control Procedures

IntelliJ IDEA makes it possible to interact with the different version control systems (VCS). Regardless of which VCS you use, certain basic operations are common to all (or almost all) of them. This section covers these basic operations and explains how to perform them from within IntelliJ IDEA:

- Adding Files to Version Control
- Browsing Contents of the Repository
- Getting Local Working Copy of the Repository
- Changing Read-Only Status of Files
- Checking in Files
- Checking Project Files Status
- Copying, Renaming and Moving Files
- Deleting Files from the Repository
- Handling Differences
- Handling Issues
- Managing Changelists
- Refreshing Status
- Reverting Local Changes
- Reverting to a Previous Version
- Shelving and Unshelving Changes
- Updating Local Information
- Using Patches
- Viewing Changes Information
- Accessing the Authentication to Server Dialog

Adding Files to Version Control

If a new file is created with IntelliJ IDEA in a directory that is already under version control, it automatically adds to the active changelist with the status Added. All you have to do, is to commit this change.

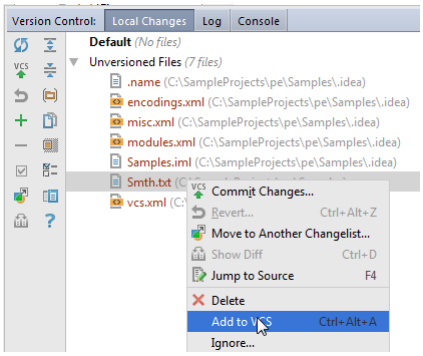
IntelliJ IDEA's behavior on adding files is configurable in the General Settings tab of the Version Control dialog.

If a new file is created outside of IntelliJ IDEA, or silent adding is disabled, you have to explicitly add it to the version control.

Another approach is VCS-specific. You can import an entire directory to your repository, provided that you have the corresponding access rights. This action is helpful for putting a whole project under version control.

To explicitly add a file to version control

1. Select file in the Project tool window.
2. On the main Version Control menu or on the context menu of the selected file, choose <VCS> | Add.
   Alternatively, use the Version Control tool window. Select the desired files in the Unversioned files changelist in the Local Changes tab, and choose Add to VCS on the context menu, or just drag it to the target changelist.



3. Select the added file in the changelist and check in (commit) changes .

Browsing Contents of the Repository

Prior to checking files out, you can browse the contents of the repository. This action is available for CVS, Git and SVN integrations.

- Browsing CVS Repository
- Browsing Subversion Repository

Browsing contents of CVS, Git and SVN repositories is always available, even when the respective VCS is not enabled in project. All you need is a valid user account.

Getting Local Working Copy of the Repository

On this page:

- Basics
- Checking out

Basics

As soon as version control support is enabled in IntelliJ IDEA, you can retrieve the data from the repository. Depending on your purpose and workflow, choose one of the following approaches:

- Check out the repository sources to a content root of an existing project . Then set up version control in the project, if it has not been done before, and put the downloaded sources under control of the VCS used.
- Check out the repository sources to a location of your choice and have IntelliJ IDEA create a project around them . This does not require any extra steps from your side. IntelliJ IDEA arranges the downloaded sources in a project itself and suggests to open it as soon as the check-out is completed.
  Upon your consent, the New Project from Existing Code Wizard starts. When the project is created, set up version control in it.

> Warning! This approach is not available for Perforce integration.

Checking out

The check-out procedure depends on the type of VCS you use. Refer to the following sections for details:

- Retrieving Files from a CVS Repository
- Retrieving Files from an SVN Repository
- Cloning a Remote Git Repository
- Cloning a Remote Mercurial Repository
- Using Perforce Integration

Changing Read-Only Status of Files

In this section:

- Basics
- Enabling explicit removal of read-only status
- Changing writable status by icon
- Example

Basics

Different version control systems have different semantics for the action of removing read-only status from a file so that you can edit it. Some systems never put read-only status on local files at all unless specifically configured to do so (i.e. the system is configured to support the file locking model).

Different version control systems use different names for this action: check out, edit, Open for Edit, or Get. Regardless of the terminology used by your VCS, if it sets read-only status on your local working files, you can remove read-only status and make files writeable from within IntelliJ IDEA, which will also take care of setting a lock on the server, or take whatever other action is required by the VCS, via the respective VCS integration.

This behavior is configurable in the Confirmation page of the Version Control settings.

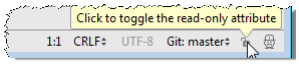
Enabling explicit removal of read-only status

To enable explicit removal of read-only status:

- In the Confirmation page of the Version Control settings , check the option Show "Clear Read-Only Status" Dialog.

Changing writable status by icon

You can make a file writable using the lock icon in the Status bar. Open the desired file in the editor, and click the lock, as shown below:



Example

Removing read-only status in Perforce looks as follows:

1. With the Show "Clear Read-Only Status" Dialog option enabled, an attempt to edit a file brings up the respective dialog box.
   If you click the radio button Using file system, the file will not be added to the default changelist. If you click radio button Using version control integration, the file is added to the default changelist.

Checking In Files

In this section:

– Basics

– Checking changed files in

Basics

Different version control systems have different semantics for the action of uploading changed files to the repository. Two common terms are check in and commit.

In those version control systems, for example, Git , that distinguish between local and remote repositories, the term commit denotes uploading changes to a local repository. Uploading changes to a remote repository is referred to as push.

Regardless of the terminology, you can perform this operation with the VCS configured for a directory from within IntelliJ IDEA.

Checking changed files in

To check in (commit) changed files, perform these general steps

1. In the Version Control tool window, select one or more files you want to check in (commit) to version control.

2. Open the Commit Changes dialog box by doing one of the following:
   – On the Version Control tool window toolbar, or on the main toolbar, click VCS.
   – Press Ctrl+K.
   – On the main menu, choose VCS | Commit Changes.

3. Review the changes to be committed in the Details pane. To do that, unfold the Details pane if it is hidden, and select the file in question in the Changed Files area.
   The Details pane shows the base version and the local copy of the selected file.

   Examine the details of each change:
   – To move to the next updated piece of code, click the Next Change button.
   – To return to the previous updated code fragment, click the Previous Change button.
   – To expand or narrow the context of an updated code fragment, position the cursor at the change in question, click the More/Less Lines button, and then specify the number of lines to be shown above and below the current code fragment.

4. Add a commit comment. As you type, IntelliJ IDEA checks the spelling and highlights words in question, provided that the Spelling code inspection is enabled.

5. Specify which actions should be performed on the files before and after submitting them to the repository.

6. Click the Submit/ Commit button to launch the Check-in Changes operations.

   Tip For Git and Mercurial. To have the changes immediately pushed to your Git or Mercurial repository, do one of the following:
   – Hover the mouse pointer over the Submit/ Commit button and select Commit and Push on the context menu.
   – From the Submit/ Commit drop-down list, select Commit and Push.

7. To save the changes as a patch in a text file, hover the mouse pointer over the Submit/ Commit button and select Create Patch on the context menu.

   Tip Alternatively, use the Submit/ Commit drop-down list to select the Create Patch item.

   In the Create Patch dialog box, that opens, configure the patch creation .

8. If any error occurs when trying to commit, IntelliJ IDEA displays an error message. For example, you might have changed a file that has been already edited by another team member, or you might run into a branch conflict. In these cases, you need to merge edits , or update your local copy . The error messages are VCS-specific.

Tip Users of JetBrains TeamCity can obtain the TeamCity plugin for IntelliJ IDEA. Among the features of this plugin is Remote Run, which enables you to create a special *personal* build that does not affect the *real* build. Your changes are built and run through your test suite. If all tests are passed, your changes are automatically committed to version control.

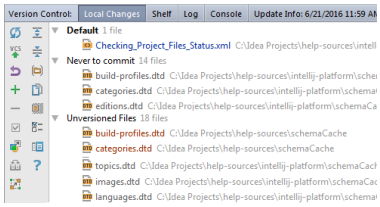
Checking Project Files Status

Apart from indicating the status of the currently opened file relative to the repository, IntelliJ IDEA provides you with an accumulated view of the project files' statuses.

To view the differences between the current state of the project files and the repository, do the following:

1. From the main menu, choose VCS | Refresh FIle Status.
2. Switch to the Version Control tool widnow and open the Local Changes tab.

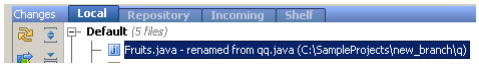
The status of each file is indicated by the color :



Copying, Renaming and Moving Files

There is no automatic renaming or moving files in the version control systems. Working with the clients, you have to manually copy a file to a new location, change its name, add to VCS and remove the old file. IntelliJ IDEA's refactoring features make it possible to easily move and rename files under version control, performing all actions involved in these functions.

In the Version Control tool window, next to the resulting files IntelliJ IDEA explicitly states: `renamed from <file name>` or: `copied from <package name>`.



To copy, move or rename a file under version control

1. Select the desired file in the Project tool window, and perform refactoring procedures, as described in the sections:
   - Copy/Clone
   - Move
   - Rename

   All added and deleted files are placed to a changelist.

2. Commit changes to the repository.

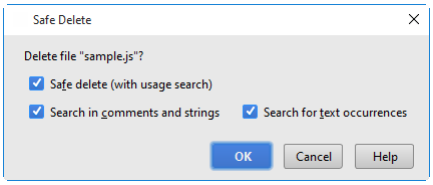
Tip You can revert refactorings, using the Undo command.

Deleting Files from the Repository

If you delete a file under version control, it still exists in the repository until deletion is committed. A deleted file is placed to the active changelist, and is displayed in grey font.

To delete a file

1. Select a file in any navigation view, and press `Delete`, or choose Delete on the context menu.
2. In the dialog that opens, you can opt to delete file without search for usages, or perform safe delete, to make sure that you are deleting an unused file, by checking Safe delete option. In this case, specify the refactoring options.

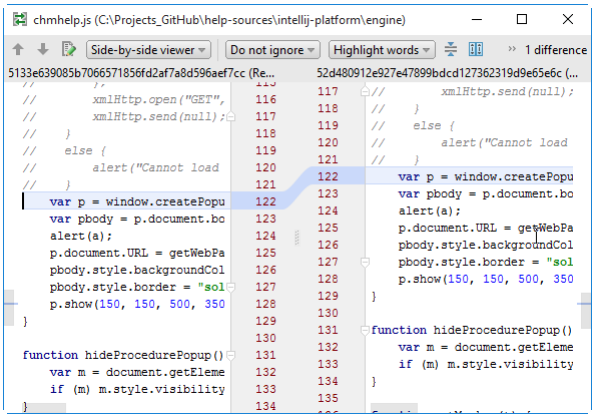


   The encountered usages are reported in the Usages Detected dialog box. You can view and correct these usages in the Safe Delete tab of the Find tool window. The deleted files are added to a changelist.

3. Commit changes to the repository.

Handling Differences

IntelliJ IDEA enables you to examine the differences between two revisions of a file, or between the current local copy and any repository version. Differences display in the Differences viewer . This window enables you to compare files and versions, navigate and search through the changes, copy and edit source code. Use the legend of the viewer to recognize modified, deleted and added lines of code.



This section describes how to:

- Compare file versions .
- Integrate contents of different versions into a file .
- Resolve conflicts between versions .
- Integrate your local copy of a project into a revision in the repository .

Comparing File Versions

VCS-specific integrations enable you to compare your local copy of a file with the versions stored in the repository.IntelliJ IDEA suggests several options:

- Compare your local copy with the repository version, to which you have last synchronized
- If somebody else has committed changes since your last update, compare your local copy with newest version
- Compare your local version with any repository version of a file .

For some VCS, it is possible to compare with a branch version. The differences display in the Differences viewer .

To compare with a repository version, to which you have last synchronized

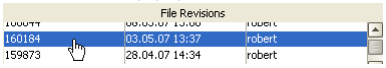
1. Select a file in the Project tool window, or open it in the editor.
2. Do one of the following:
   - On the main VCS menu, or on the context menu of a file, choose <VCS> | Compare with the same repository version.
   - Select a file in the Local Changes tab of the Version Control tool window, and choose Show Diff on the context menu.

To compare with the latest repository version

1. Select a file, or open it in the editor.
2. On the main VCS menu, or on the context menu of a file, choose <VCS> | Compare with the latest repository version.

To compare with the specified version of a file

1. Select a file, or open it in the editor.
2. On the main VCS menu, or on the context menu of a file, choose <VCS> | Compare with.
3. In the File Revision pop-up window, click the version to compare:

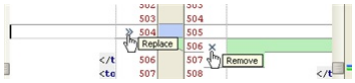


Tip Alternatively, use the History view of a file. Select the desired version, and choose Compare with Local on its context menu, or click on the toolbar.

Tip You can explore changes to binary files same way, as you do it for textual files. For example, use this feature to watch the changes made to images.

Integrating Differences

You might need to integrate contents of the different versions of a file, for example integrate changes of a certain branch into your local copy.



To integrate changes into your working copy

1. Select the desired file and compare it with a version .
2. In the Differences Viewer, use the *chevron* symbol for any block of lines that existed in the read-only copy and were changed or deleted in the local copy; and *x* symbol for the lines that were added in the local copy. Click these symbols to apply changes to the current local version of the file.

Integrating Project

You might need to integrate your local version of a project into a certain revision of that project in the repository. For this purpose IntelliJ IDEA provides Integrate Project command.

Integrating project is only available for the projects that contain directories associated with Perforce or Subversion. If neither of these two VCS is involved in project, the Integrate Project command is disabled in the menu.

To integrate different sources into one project

1. On the main menu, choose VCS | Integrate Project. The Integrate dialog appears.
2. Select Perforce or Subversion tab, and configure merge options as required. Refer to the respective procedural topics for details.

Resolving Conflicts

Depending on the type of your version control system, conflicts arise in different situations.

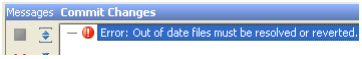
Non-Distributed Version Control Systems

Working in a team, you might come across a situation, when somebody commits changes to a file you are currently working on. Conflicts can also arise on merging branches, applying patches or unshelving changes. If the changes are not overlapping, the files are merged automatically, but when there are changes to the same lines, you will need to resolve such conflict yourself.

When you try to edit a file that has a newer version on the server, IntelliJ IDEA informs you about that, showing a message pop-up window in the editor:

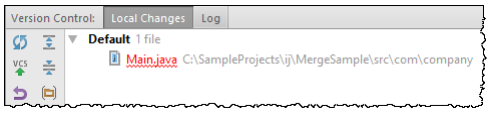
In this case you should update your local version prior to changing the file, or merge changes.

If you try to commit a file that has a newer repository version, an error is reported in the Messages tool window:



The commit behavior is regulated by the Create changelist on failed commit drop-down list in the Confirmation page of the Settings / Preferences dialog.

If you synchronize a file, that already has local changes, with a newer repository version, committed by somebody else, a conflict occurs. The conflicting file gets status Merged with conflicts. The file remains in the same changelist in the Local Changes tab of the Version Control tool window, but its name is highlighted red. If the file is currently opened in the editor, the file name on the tab header is also highlighted red.

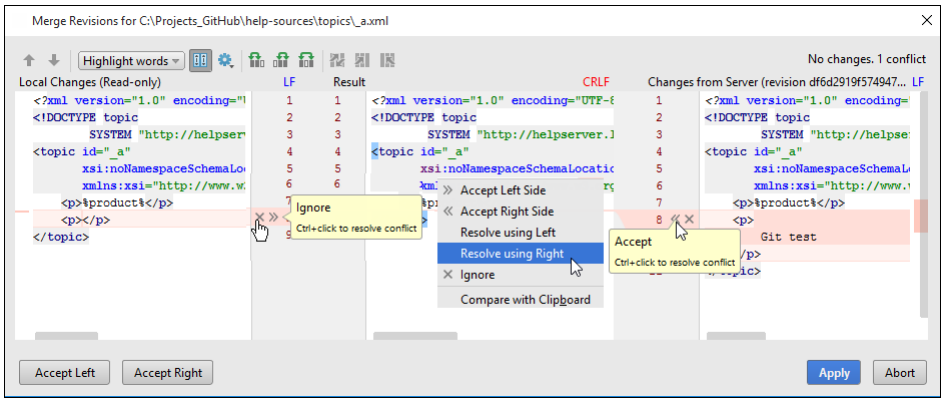


Distributed Version Control Systems

Under Git, conflicts arise during merge , rebase , cherry-pick , unstash , or apply patch .

Conflict Resolution Tool

IntelliJ IDEA suggests a tool for resolving conflicts locally. This tool consists of three panes. The left pane shows the read-only local copy; the right pane shows the read-only version checked in to the repository. The central pane shows a fully-functional editor with the results of merging and conflict resolving. Initially, the contents of this pane is the same as the base revision of the file, that is, the revision from which both conflicting versions derived.



To resolve conflicts, follow these general steps

1. Select the conflicting file in the Version Control, Update Version Control, or Project tool window, or open it in the editor.
2. On the context menu, under the node of your VCS integration, choose the VCS-specific merge command.
3. Perform edit in the merge tool to apply or discard changes, as described in the procedures below.

To apply or discard a change

1. In the left or right pane, select the change to be merged. Use arrow and/or find buttons on the toolbar to navigate to the desired location.
2. Accept or reject changes, clicking the chevron or X button respectively.
3. Click Apply.

To apply all non-conflicting changes

– On the toolbar of the merge tool, click . It is useful to automatically accept any changes that are not in conflict, and spend your time working on conflicting changes.

Running IntelliJ IDEA as a Diff or Merge Command Line Tool

In this topic:

- Overview
- Enabling invocation of IntelliJ IDEA operations from the command line
- Comparing files using IntelliJ IDEA as a command line tool
- Merging files using IntelliJ IDEA as a command line tool
  - Passing three arguments to merge tool

Overview

Besides using IntelliJ IDEA as an Integrated Development Environment, you can use it as a command line tool for comparing and merging files.

IntelliJ IDEA executable is platform-dependent:

- Windows: `IntelliJIdeaXX\bin\idea.exe` / `IdeaICXX\bin\idea.exe` or `IntelliJIdeaXX\bin\idea.bat` / `IdeaICXX\bin\idea.bat`
- UNIX: `IntelliJIdeaXX/bin/idea.sh` / `IdeaICXX/bin/idea.sh`
- OS X: `/Applications/IntelliJIdeaXX.app/Contents/MacOS/idea` / `/Applications/IdeaICXX.app/Contents/MacOS/idea`
  To add the launcher to your path, add its containing directory `/Applications/IntelliJ IDEA.app/Contents/MacOS` .

However, for OS X and UNIX, one should create a wrapper script, since this helps avoid some drawbacks related to the usage of IntelliJ IDEA launcher.

Enabling invocation of IntelliJ IDEA operations from the command line

For OS X and UNIX platforms, we recommend creating the wrapper script, or the command line launcher to integrate IntelliJ IDEA with your shell. Then, you need to ensure that the created launcher script is within the search path of your shell.

On Windows, we recommend to add the path to IntelliJ IDEA to the environment variable `Path` . Everything is done outside of IntelliJ IDEA, with the IntelliJ IDEA executable.

Note that if you have specified location of the IntelliJ IDEA executable as a `Path` environment variable, the command will work no matter which directory you are currently in.

To enable invoking IntelliJ IDEA operations from the command line, follow these steps

- On OS X or UNIX:
  1. Make sure IntelliJ IDEA is running.
  2. On the main menu, choose Tools | Create Command-line Launcher. The dialog box Create Launcher Script opens, with the suggested path and name of the launcher script. You can accept default, or specify your own path.
     Make notice of it, as you'll need it later.
  3. Outside of IntelliJ IDEA, add the path and name of the launcher script to your path.

- On Windows:
  - Specify the location of the IntelliJ IDEA executable in the `Path` system environment variable. In this case, you will be able to invoke the IntelliJ IDEA executable and other IntelliJ IDEA commands from any directory.

Comparing files using IntelliJ IDEA as a command line tool

To compare two files using IntelliJ IDEA as a diff command line tool

1. Enable invoking IntelliJ IDEA operations from the command line .
2. Type the following command at the command line prompt:

```
<IntelliJ IDEA launcher(Windows) or wrapper script (MacOS or UNIX)> diff <path to file1> <path to file2>
```

   where `file1` is your local copy, `file2` is the repository version.

For example:

```
idea diff README.md.bak README.md
```

Merging files using IntelliJ IDEA as a command line tool

Most often you need to merge three versions of the same file: your local version, the version in the repository or in the upstream, and the base revision, which is the origin for the two diverged versions.

To merge files using IntelliJ IDEA as a command line tool

1. Enable invoking IntelliJ IDEA operations from the command line .
2. Type the following command at the command line prompt:

```
<IntelliJ IDEA launcher(Windows) or wrapper script (MacOS or UNIX)>
merge <path to file1> <path to file2> <path to file3> <path to output>
```

where `file1` is your local copy, `file2` is the repository version, `file3` is the base revision for `file1` and `file2`, and `output` is the file to save the merge results in (optional).

Passing three arguments to merge tool

It is possible to pass just three arguments to the merge tool:

`<path to file1> <path to file2> <path to output>`.

In this case, the contents of the output will be taken as the base revision:

```
<IntelliJ IDEA launcher> merge <path to file1> <path to file2> <path to output> <path to output>
```

See the example in this blog to learn how to use IntelliJ IDEA diff and merge tool with Git.

Handling Issues

With IntelliJ IDEA, you can connect your check-in comments with the bug tracker or any issues data base and navigate from committed changes to the issues related to these changes.

To enable this navigation, you need to specify a so called issue navigation pattern, which means:

1. Figure out an issue ID pattern, that is, a format according to which you will reference issues in commit messages, and define this issue pattern using a regular expression.
2. Define the link to the referenced issue by combining the URL address of your tracking system and a regular expression to identify the issue ID.

In other words, an issue navigation pattern maps an issue ID pattern in commit messages and URL addresses of referenced issues. As soon as IntelliJ IDEA encounters a match to the issue ID pattern in a commit message, the match is displayed as a link in the Changes and Version Control tool windows. If you mention several issues, all of them will show up as links. Clicking such link opens the matching issue in the browser according to the defined link.

On this page:

– Example
– Enabling navigation from commit messages to issues related to them
– Navigating from a commit message to the related issues

Example

Issue ID pattern	The regular expression that defines the format in which issues are referenced in commit messages. `[A-Z]+\-\d+` This regular expressions matches all character strings that consist of two substrings separated by an n-dash character: 1. Substring 1: An unlimited number of upper case alphabetic characters. 2. Substring 2: An unlimited number of digital characters.
Issue link pattern	A combination of the URL address of your issue tracking system and a regular expression that identifies issues in it. `http://mytracker/issue/$0` Here `$0` indicates a back reference to the entire match. This means that as soon as IntelliJ IDEA detects a match in a commit message, it is added to the URL address of the tracker as is.
Matching issue ID	IntelliJ IDEA detects the following reference to an issue in the commit message of interest: `MYPROJECT-110`
Composed issue link	In accordance with the above issue navigation pattern, the detected matching reference is added to the URL of the tracker as is, so the link to the referenced issue is composed as follows: `http://mytracker/issue/MYPROJECT-110`

To enable navigating from commit messages to issues related to them

1. Open the Settings dialog box , and click Issue Navigation under the Version Control node.
2. In the Issue Navigation dialog box that opens, configure a list of issue navigation patterns by setting correspondence between issue patterns in commit messages and URL addresses of referenced issues.
   – If you are using JIRA or our bug tracking system YouTrack , click the Add JIRA pattern or Add YouTrack Pattern respectively, and type the URL to the installation of bug tracking system in question.

     IntelliJ IDEA adds the regular expression that defines such pattern automatically.
   – For other issue tracking systems, click the Add button to create a new entry or select an existing entry and click the Edit button. In the Add Issue Navigation Link dialog box that opens, specify the following:
     1. The regular expression that defines the issue pattern in a commit message.
     2. The replacement expression that defines the URL to access the corresponding referenced issue.
   – To remove an issue navigation pattern, select it in the list and click Remove .

To navigate from a commit message to the related issues

1. Open one of the following views:
   – Local Changes, Incoming, or Log tab of the Version Control tool window.
   – History tab of the Version Control tool window.

- Changes Browser .

2. Find the commit of interest and click the hyperlink to the related issue.

Managing Changelists

This section describes how to:

- Create , delete and rename changelists.
- Assign active changelists .
- Group items in a changelist .
- Move files between changelists .
- Jump from an item in a changelist to the corresponding source code in the editor .

Assigning an Active Changelist

Active changelist is the one to which the changed files are added automatically. The name of the active changelist is highlighted in bold font.

To assign an active changelist

1. Select a changelist in the Version Control tool window.
2. On the context menu of the selected changelist, click Set Active Changelist.

To create a new changelist

1. In the toolbar of the Version Control tool window, click + button.

   Tip You can also use one of these alternatives:
   - Right-click anywhere in the Local Changes tab of the Version Control tool window and choose New Changelist on the context menu.
   - Press `Alt+Insert`.

2. In the New Changelist dialog, specify the name of the new changelist, and optional comment.

3. Click OK.

Deleting a Changelist

When a changelist is deleted, all changes are moved to the active changelist.

To delete a changelist

1. In the Version Control tool window , select a changelist to be deleted.
2. In the toolbar of the Version Control tool window, click —. Alternatively, right-click the changelist node and choose Delete Changelist on the context menu, or just press `Delete` key.
3. If the changelist is not empty, you are prompted to confirm deletion and move uncommitted items to the active changelist. If you attempt to delete an active changelist, you are prompted to specify another
   If Perforce is used for a certain directory, deleting the default changelist is not allowed.

Grouping Changelist Items by Folder

Within each node of the Version Control tool window , you can display the modified files as a flat list, or as a directory tree.

To toggle grouping items by directories, do one of the following

- On the toolbar of the Version Control tool window, click .
- Use `Ctrl+P` keyboard shortcut.

To move items between changelists in the Version Control tool window

1. In the Version Control tool window, select one or more desired items in a changelist. Use the `Ctrl` and `Shift` keys for multiple selection.

2. Choose Move to Another Changelist on the context menu of the selection.
   You can also use one of these alternatives:
   - Click the Move to Another Changelist button on the toolbar of the tool window.
   - Press `F6`.
   - Drag the selected items to the target changelist.

3. In the Choose Changelist dialog box, specify the changelist to move the selected items to:
   - If the target changelist exists, click the Existing Changelist option and select the desired changelist from the drop-down list.
   - To create a changelist, click the New Changelist option, type the name of the new changelist, and optionally provide a description.

To navigate from an entry in the changelist to the source code

1. In the Version Control Tool Window, expand a changelist and select the desired entry.
2. On the context menu of the selection, choose Jump to Source, or press F4.

To rename a changelist

1. Select a changelist in the Version Control Tool Window.
2. On the context menu, choose Edit Changelist.
3. In the Edit Changelist dialog, specify the new name and optional description, and click OK, or use the `Shift+F6` keyboard shortcut..

Refreshing Status

This feature is helpful if a server, that holds the sources of the project files, is down. The statuses of files and directories become irrelevant, and you are unable to work with the local copy of the project. To avoid such situation, you need to refresh the status of your source files.

When this command is applied, IntelliJ IDEA refreshes the status of each file, no matter whether the file has been changed from IntelliJ IDEA itself or using any other application.

When working under Perforce control, you can run refresh in two modes:

- Standard Refresh takes into consideration only the changes made through the IntelliJ IDEA integration with Perforce. This improves performance because does not require connecting to the server. However, this approach does not let you know about the changes made outside IntelliJ IDEA, for example, right through the `p4v client` application.
- Force Refresh considers all the changes made to project, both from IntelliJ IDEA and from any other application, for example, right through `p4v client`.

The status of a file is refreshed only in accordance with the changes in your local workspace. Any changes on the server can be reflected only through synchronization .

To refresh the status of files in your project, do one of the following

- On the main menu, choose VCS | Refresh File Status.
- In the Version Control tool window, press `Ctrl+F5`.
- In the Version Control tool window, click the Refresh toolbar button .
- For Perforce integration, do one of the following:
  - To run Standard Refresh, click the Refresh toolbar button or press `Ctrl+F5`.
  - To run Force Refresh, click the Force Refresh toolbar button .

Reverting Local Changes

When you modify, add, or delete files, which are under version control, you are always able to revert such changes, rolling back the file's contents to what it was before the last successful update, check out, or commit.

The exact name of the command (revert or roll back), and the type of the action performed when you revert changes, depend on the file status and VCS used for a particular directory.

To revert local changes, do one of the following

- In the Local Changes tab of the Version Control tool window, select one or more items in the relevant changelist , then choose Revert on the context menu of the selection or click the Revert button on the toolbar of the Version Control tool window.
- Select the file to be reverted and choose the relevant VCS-specific revert (roll back) command on the VCS | <VCS> menu.

Reverting to a Previous Version

You can restore any previous version of a file, using the History view of a file. So doing, the current content of the file is replaced with the copy of the older content. After such operation, you have to commit the file to bring the repository up to date.

The exact name of the command (revert or roll back) depends on the specific VCS.

To revert a file to its previous version

1. Select the desired file in the Project tool window, and open its history .
2. In the History tab, select the desired revision.
3. On the context menu of the selected entry, choose Get, or click the button on the toolbar.

Shelving and Unshelving Changes

Shelving is temporarily storing not yet committed changes in a dedicated shelf .

Unshelving is returning postponed changes from a shelf to a pending changelist.

With IntelliJ IDEA, you can shelve and unshelve both separate files and entire changelists or shelves.

Unshelved changes can be filtered out from the view or removed from the shelf. Once shelved, a change can be applied as many times as you need by unshelving and subsequently restoring it on the shelf.

In the Git integration , in addition to shelving and unshelving "stashing" and "unstashing" are supported respectively. These features have much in common, the only difference is in the way patches are generated and applied.

- Patches with stashed changes are generated by Git itself. To apply them later, you do not need IntelliJ IDEA.
- Patches with shelved changes are generated by IntelliJ IDEA. Normally, they are also applied through the IDE. Applying shelved changes outside IntelliJ IDEA is also possible but requires additional steps.

In this section:

- Shelving Changes
- Unshelving Changes
- Restoring Unshelved Changes
- Filtering Out and Removing Unshelved Changes

Shelving Changes

On this page:

– Basics
– Shelving changes
– Shelving base revision automatically

Basics

Shelving is temporarily storing not yet committed changes in a dedicated shelf . This may be helpful, for example, when you need to switch to another urgent task, but you are short of time to bring your current code to a certain condition.

With IntelliJ IDEA, you can shelve both entire changelists or separate files.

Once shelved, a change can be applied as many times as you need by unshelving and subsequently restoring it on the shelf.

When using Git or Mercurial integration, it may be useful to have IntelliJ IDEA always shelve the base revisions of files that are under Git or Mercurial version control.

By default, IntelliJ IDEA always "remembers" the last commit hash. However, this information is not sufficient if the history has been changed since the last commit as a result of running the Rebase operation. In this case, having a copy of the base revision may help.

Shelving changes

1. In the Local Changes tab of the Version Control Tool Window tool window, select the files or changelist to put to a shelf . On the main Version Control menu or on the context menu of the selection, choose Shelve changes.
2. In the Shelve Changes dialog box, review the list of changed files and make sure that the files to be put to a shelf are checked out.

   Tip You can specify the desired changelist immediately in the dialog box: from the Changelist combo box, select the desired changelist.
3. In the Commit Message text box, type the comment to identify the shelf where the changes will be stored.
4. Click Shelve changes.

Shelving base revision automatically

1. Open the Settings dialog box, and click Version Control.
2. On the Version Control page, select the Store on shelf base revision texts of files under DVCS check box.

Unshelving Changes

On this page:

- Basics
- Unshelving Changelists and Files
- Applying External Patches

Basics

Unshelving is moving postponed changes from a shelf to a pending changelist.

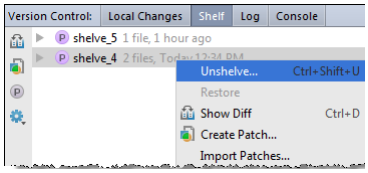
With IntelliJ IDEA, you can unshelve both entire shelved changelists and separate files.

Unshelved changes can be filtered out from the view or removed from the shelf. Once shelved, a change can be applied as many times as you need by unshelving and subsequently restoring it on the shelf.

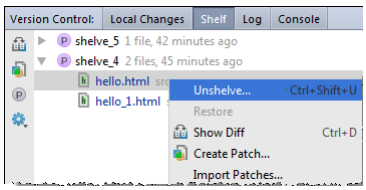
You can also import patches created inside or outside IntelliJ IDEA and apply them as shelved changes.

Unshelving Changelists and Files

1. In the Shelf tab of the Version Control tool window , select the changes to unshelve:
   - To unshelve an entire changelist, just select its name.



   - To unshelve the changes from a specific file, expand the relevant changelist node and select the desired file.



2. On the context menu of the selection, choose Unshelve.
3. In the Unshelve Changes Dialog that opens, specify the changelist you want to restore the unshelved changes to.
   - To add the changes to an existing changelist, choose Existing Changelist and select the relevant changelist from the drop-dow list.
   - To create a new changelist and restore the changes from the shelf in it, choose New Changelist.
     - In the Name text box, type the name of the changelist to be created.
     - In the Comment text box, type an optional description of the new changelist.
     - To have IntelliJ IDEA automatically give the active status to the new change list immediately after the changes are restored in it, select the Make this changelist active check box. When this check box is cleared, the current active changelist remains active. See Changelist for details.
     - To have IntelliJ IDEA preserve the context of the task associated with the new changelist on its deactivation and restore the context when the changelist becomes active, select the Track context: check box. See Managing Tasks and Context for details.
4. To have IntelliJ IDEA still display already unshelved changes in the Shelf tab so that you can apply them once more if necessary, clear the Remove successfully applied files from the shelf check box.
   When this check box is selected, the changes are not displayed in the Shelf tab after they are unshelved.
5. Click OK to start the process.
   If conflicts occur between the patched version and the current version, resolve them as described in Resolving Conflicts . Select the changes in both versions to be merged to the resulting file and click Apply.

Applying External Patches

You can also import patches created inside or outside IntelliJ IDEA and apply them as shelved changes.

1. In the Shelf tab of the Version Control tool window, choose Import Patches from the context menu.
2. In the dialog box that opens, select the patch file to apply. The selected patch appears in the Shelf tab as a shelf.
3. Select the newly added shelf with the patch and choose Unshelve Changes from the context menu of the selection.

Restoring Unshelved Changes

With IntelliJ IDEA, you can apply changes as many times as you need by restoring them to the shelf upon unshelving. You can restore both separate files and entire shelves. Unshelved changes are available for re-use until they are explicitly removed, see Filtering Out and Removing Unshelved Changes .

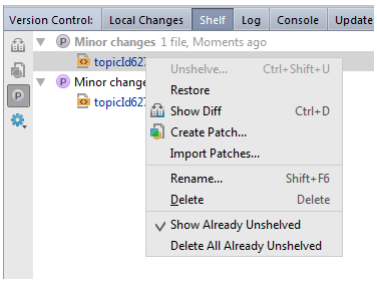
By default, upon unshelving, a change is no longer shown on the list of available shelved changes, therefore you first need to have IntelliJ IDEA display it in the Shelf tab.

To restore unshelved changes

1. In the Shelf tab of the Version Control tool window, click the toolbar button or choose the Show Already Unshelved option on the context menu. All previously applied shelves are again on the list and are available for unshelving.

2. Select the desired files or shelf to restore.

3. On the context menu of the selection, choose Restore.



Filtering Out and Removing Unshelved Changes

Most shelved changes are applied only once. By default, upon unshelving a change is no longer shown on the list of available shelved changes. This helps you keep your list of available shelved changes more easier to use. However you can return an applied shelf to the list it at any time, restore and unshelve it again as many times as necessary.

Shelves that are not likely to be applied in the future anymore, can be permanently removed from the list.

On this page:

- Removing a single change
- Returning unshelved changes to the list
- Hiding unshelved changes
- Removing unshelved changes permanently

Removing a single change

You can permanently delete any change no matter whether it has been already unshelved or not.

- To remove a shelved change permanently, select it in the list and choose Delete on the context menu of the selection.

Returning unshelved changes to the list

- In the Shelf tab of the Version Control tool window, choose Show Already Unshelved on the context menu.

Hiding unshelved changes

- In the Shelf tab of the Version Control tool window, choose Hide Already Unshelved on the context menu.

Removing unshelved changes permanently

1. In the Shelf tab of the Version Control tool window, choose Show Already Unshelved on the context menu.
2. Select the desired file or changelist.
3. On the context menu of the selection, choose Delete All Already Unshelved.

Updating Local Information

The Update command enables you to synchronize the contents of your local files with the repository. You can invoke this command on:

- Single or multiple files and directories
- An entire project

Depending on the updating options, the update procedure may take place silently. If all files are up-to-date, you will be notified about that. Otherwise, the Update Info tab opens in the Version Control tool window where you can group the information as required.

On this page:

- Updating files and folders
- Updating a project
- Grouping update information by packages or changelists

Updating files and folders

1. Select one or more files and folders to be updated in any navigation view (for example, in the Project tool window .
2. On the main Version Control menu, or on the context menu of the selection, choose <VCS>| Update.
3. In the Update dialog specify the update options, which are different for the supported version control systems, and click OK.

Updating a project

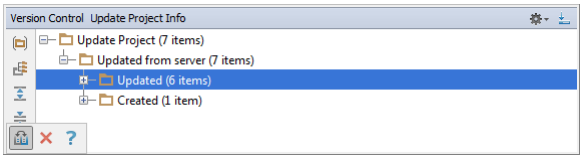
1. Do one of the following:
   - On the main menu, choose VCS | Update project
   - Press `Ctrl+T`.
   - On the main menu, click .
2. In the Update dialog, click the tab for your version control system.
3. Specify the update options, which are different for the supported version control systems and click OK.

Grouping update information by packages or changelists

To group the update information by packages or changelists, use the Group By Package and Group By Changelist buttons on the toolbar of the Update Info tab.

Note the difference in the appearance of the tab:

- Grouped by packages



- Grouped by changelists

Grouping by changelists is not available if the project is under Git or Mercurial control.

Using Patches

This section describes how to:

- Create patches .
- Apply patches .

Applying Patches

Postponed changes stored in a patch file can be applied to the target file or directory later. If the source code was edited after creating the patch, conflicts may arise. IntelliJ IDEA suggests a handy way to resolve such conflicts and merge the patch with the changes.

When a patch is opened, IntelliJ IDEA detects files with the same name as the modified files. For each detected file, IntelliJ IDEA compares the path to it relative to the base directory, with the path from the patch, and chooses the closest match. If no matching path is found, the file is considered to be located in the project base directory and is highlighted in red.

You can apply changes to files stored in different locations from those specified in the patch by mapping an arbitrary directory as the base one, or stripping off the leading directories.

To apply a patch

1. On the main menu, choose VCS | Apply patch.
2. In the Apply Patch dialog box that opens, specify the fully qualified name of the patch file. Type the name manually or click the Browse button and locate the desired patch file in the Select Patch File dialog box.

   You can also drag and drop a file or an email attachment to the Apply Patch dialog, and it will be selected automatically.
3. Configure the patch presentation layout. To have changes shown in a flat view, press the Group by Directory toolbar button .

   Release the button to have changes shown in a directory tree view.
4. To have a change applied, select the check box next to it.
5. To have a change applied to a modified file that has been moved to another location, specify the new file location.
   - To map another base directory, select the desired file, directory, or a group files/directories and click the Map base directory toolbar button . In the dialog that opens choose the directory relative to which file names will be interpreted.
   - To remove leading directories from the path, click the Strip Directory toolbar button as many times as many leading directories you need to strip. The number of removed slashes is indicated in square brackets.
   - To revert the last strip directory action, click the Restore Directory toolbar button . Click the button as many times as many previously stripped leading directories you need to restore.
   - To revert all the strip directory actions in the selection, click the Reset Directories toolbar button .
   - To have all the leading directories stripped and have the changes applied to the file with the specified name in the base directory, click the Remove Directories toolbar button .
6. To view the differences and possible conflicts between your local working copy, the repository version, and the patch in the Differences viewer for files , select the desired change and click the Show Differences button .
7. To resolve conflicts between the patched and the current versions, if any, in both versions select the changes to be merged to the resulting file, and then click Apply.
   Note that you apply a patch that contains a lot of files and causes numerous conflicts, you can cancel applying the patch by clicking Abort, and then select whether you want to abort, skip or cancel the remaining conflicts.

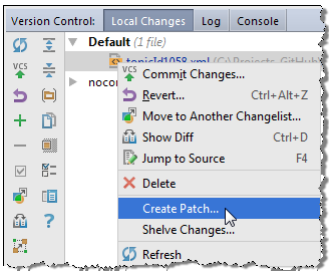
Creating Patches

IntelliJ IDEA suggests two ways of creating patches:

- On the base of revisions, either local or committed to the repository
- On the base of revisions stored in the local history.

To create a patch file

1. In the Local Changes tab or the Repository tab of the Version Control tool window, select a change or changelist you want to create a patch for.
2. On the main Version Control menu or on the context menu of the selection, choose Create patch.



3. In the Create Patch dialog box that opens, review the list of changed files, and make sure that the files to be included in the patch are selected.

   Tip You can specify the desired changelist immediately in the Create Patch dialog box: click Changelist combo box in the upper right corner, and select the desired changelist.

4. Add a commit comment.

   Tip As you type, IntelliJ IDEA checks the spelling and highlights erroneous words.

   This functionality is available if the 'Spelling' code inspection is enabled.

5. Click Create patch.

You can also create patch on the base of your local history. To do that, open the local history view for the desired directory, file or code fragment, as described in the section Using Local History , right-click the desired revision, and choose the Create Patch command on the context menu, or click the create patch button on the toolbar.

Viewing Changes Information

IntelliJ IDEA provides various ways to keep track of the changes that you and your teammates introduce to the source code. You can:

- View local changes: use the editor where you create your source files to view, explore and navigate through your local changes within a single file.
  Use the Local Changes tab of the Version Control tool window to examine local changes throughout the entire project.
- View changes made by other team members:
  Use the Committed and Incoming tabs to view the changes committed to the repository by the other team members .
- Viewing history:
  Browse through the changes , filtering the range you are interested in.

  Explore history of a file in a dedicated view.

  Show annotated view of a file to drill deeper into the file history and find changes information for each line of code.

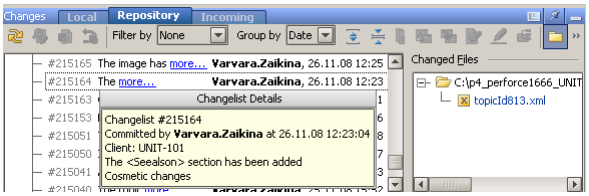
Browsing Changes

Browsing changes feature helps you see all changes during the history cache period, or confined to a certain period, user, or changelist .

All changes for the history cache period are displayed in the Repository tab of the Version Control tool window, Changelists pane.

To browse all changes in the history cache

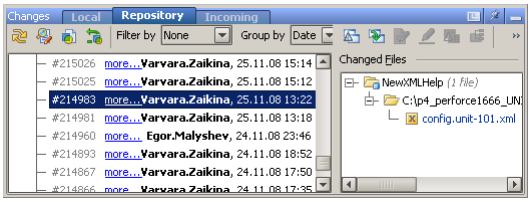
1. In the Version Control tool window, switch to the Repository tab.
2. Click the refresh button on the toolbar, or choose Refresh on the context menu.



To browse changes to files or folders

1. Select the desired file or folder in the Project window.
2. On the Version Control menu or on the context menu of the selection, choose <VCS> | Browse Changes . The Change Search Criteria dialog box is opened.
3. Specify the search criteria (type the necessary filter value or press the ellipsis button to set the value for Author, Date, and Change filters) and click OK.
   If you do not specify any filtering criteria, you will be prompted either to show recent changes only, or view all history of the entire project.

   Setting too wide filter values for a large project might require considerable time for the changes to be found.



Using Change Markers to View and Navigate Through Changes in the Editor

As you modify a file that is under version control, all changes are highlighted in the editor with the change markers that appear in the left gutter next to the modified lines, and show changes introduced since the most recent synchronization with the repository. On committing a file to the repository, the change markers disappear.

The changes you introduce to the text are color-coded:

- line added.
- line changed.
- line deleted.

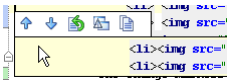
A special toolbar is associated with each change marker, enabling you to perform the following operations:

- Navigate between changes using the and buttons in the change markers or the `Ctrl+Shift+Alt+Down` or `Ctrl+Shift+Alt+Up` keyboard shortcuts. See Navigating between Changes for details.
- Compare the current version with the last committed one using .
- Revert changes using .
- Put the previous version of the changed lines to the clipboard using .

To show the change marker toolbar

1. Point to the desired change marker. The mouse pointer changes its shape:

2. Click the change marker. The toolbar and the previous contents of the line are displayed:



Viewing Details of Changes

Besides navigating through your local changes within a file in the editor, you can view and examine these changes compared to the base revision of the file.

This functionality is provided through a dedicated Change Details pane in the Local Changes tab of the Version Control tool window.

The pane consists of two areas:

- Area (1) shows the affected code as it was in the base revision.
- Area (2) shows the affected code as it is after the change is introduced.

In each area, IntelliJ IDEA numbers both changes themselves (4) and the lines (3) affected by them. The changes introduced to the code are color-coded:

- line changed (5).
- line added (6).
- line deleted (7).

A special toolbar (8) enables you to perform the following operations:

- Navigate between changes using the Next Change and Previous Change buttons.
- Expand or narrow the context of a change using the More/Less Lines button.
- Configure the appearance of the pane using the Settings button.

In this section:

- Opening the dedicated Changes Details pane
- Examining details of changes in a file
- Configuring the appearance of the pane
- Closing the Change Details pane

To open the dedicated Change Details pane

1. Open the Version Control tool window (`Alt+9`), and then switch to the Local Changes tab.
2. Click the Change Details toggle button on the toolbar. The Change Details pane opens.

Note that rhe Change Details toggle button remains pressed until the pane is closed.

To examine details of changes in a file

1. Open the Version Control tool window (`Alt+9`), and then switch to the Local tab.
2. Select the file in question, and then open the Change Details pane, if it is not opened. The base revision and the local version of the selected file are opened with the differences numbered and highlighted.
3. Examine the details of each change:
   - To move to the next updated piece of code, click the Next Change button.
   - To return to the previous updated code fragment, click the Previous Change button.
   - To expand or narrow the context of an updated code fragment, position the cursor at the change in question, click the More/Less Lines button, and then specify the number of lines to be shown above and below the current code fragment.

To configure the appearance of the Change Details pane

1. Click the Settings button on the toolbar of the pane.
2. On the context menu, that opens, specify how you want the pane split:
   - To have the pane split horizontally, so the base revision is shown in the upper part and the locally updated version is shown in the bottom part of the pane, select the Top/Bottom option.
   - To have the pane split vertically, so so the base revision is shown in the left-hand part and the locally updated version is shown in the right-hand part of the pane, select the Left/Right option.
3. To have the soft wraps (or word wraps) used, select the Use soft wraps option.

To close the Change Details pane

- Click the pressed Change Details toggle button on the toolbar.

Viewing Changes History for a File or Selection

VCS integrations of IntelliJ IDEA enable you to inspect the changes made to files, or even fragments of source code. Show History and Show History for Selection commands are available on the main VCS menu, and on the context menus of the files, under the respective VCS nodes.

Results of the Show History command are displayed in a dedicated tab of the Version Control Tool Window . For the different version control systems, the set of toolbar buttons in the History tab may slightly vary. Results of the Show History for Selection command are displayed in a dedicated window, in the form of a differences viewer.

Viewing the History of a File

1. Do one of the following:
   - Open the file in the editor. Then on the main VCS menu or on the context menu of the editor tab, choose <directory VCS> | Show History.
   - In the Project tool window, select the file in question and choose <directory VCS> | Show History on the context menu of the file.

   The History tab for this file appears in the Version Control tool window, the name of the file is shown on the title bar of the tab.
   File revisions are shown in regular font. The revision that currently exists in your working copy, is shown in bold font.
2. Click the columns to filter the list of changes you have made to the selected file.
3. Click the relevant buttons on the toolbar to compare with local version, get the selected version from VCS, annotate a file, refresh the revisions list, and show the list as a tree.
4. To view the other files updated in a revision, select the revision in question and choose Show All Affected Files on the context menu. The Paths Affected in Revision dialog box opens showing a list of all the files that were modified in the selected revision.

Viewing the History of a Selection

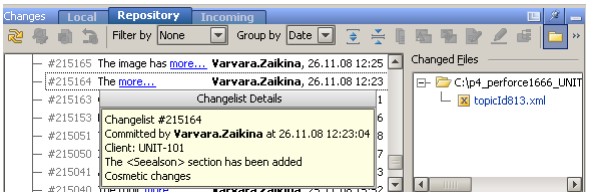
1. In the editor, select the fragment of source code to view the history for.
2. Choose <directory VCS> | Show History for Selection on the main VCS menu, or on the context menu of the selection.

Viewing Changes Made by Other Team Members

Use the Repository and Incoming tabs of the Version Control tool window to view changes introduced by the team. Repository tab shows all changes that you and your colleagues have committed to the repository; the Incoming tab shows only those committed changes, which are not yet taken to your local working copy. IntelliJ IDEA informs you about new changes in the repository by the icon on the status bar.

To view all changes committed to the repository

1. In the Version Control tool window, select the Repository tab.
2. Click refresh button on the toolbar, or choose Refresh on the context menu.



To view incoming changes

1. In the Version Control tool window, select the Incoming tab.
2. Click refresh button on the toolbar, or choose Refresh on the context menu. Alternatively, click the incoming changelists icon in the status bar:

Viewing File Status

IntelliJ IDEA provides facilities for monitoring the status of project files relative to the repository. The status of a file reflects which operations have been performed on the file locally since the last synchronization with the repository. Using this functionality requires the following prerequisites:

- Integration with a version control system is enabled.
- The entire project or its specific directories are associated with this version control system.

You can monitor the status of a project in the following ways:

- In any interface component by the color used to display the name of a file.
- In the Version Control tool window, Local tab, which by default shows files grouped according to their status. The name of nodes depend on the VCS you are using.

  Tip To have files grouped by directories, click the icon.

- For Git and CVS integrations, in the Version Control tool window, Check Project Status Info tab, via the Check Project Status command.

Working with Annotations

In this topic:

- Basics
- Showing and hiding annotations
- Configuring the amount of information shown in the annotations pane
- Annotating previous revisions
- Viewing differences between revisions
- Navigating to log

Basics

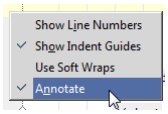
Annotation is a form of file presentation that shows detailed information for each line of code. In particular, for each line you can see the version from which this line originated, the user ID of the person who committed this line, and the commit date. The annotated view helps you find out who did what and when, and trace back the changes.

Annotating lines of code is available for ClearCase, TFS, Mercurial, Git, CVS, Perforce and Subversion.

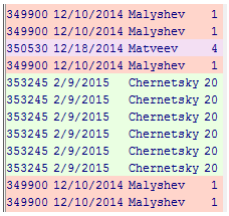
The Annotate command appears in the VCS-specific nodes of the Version Control menu, file context menus, and the File History view. This command toggles between the plain and the annotated view of a file.

Showing and hiding annotations

1. Open the desired file in the editor.
2. To show annotations, right-click the left gutter, and select Annotate:



When annotations are enabled, the left gutter looks similar to the following example:



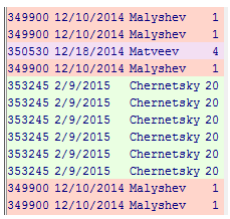
The annotations pane provides the following information:

- The number of the changelist within which the annotated change was checked in.
- The date when the annotated change was checked in.
- The name of the user who checked in the annotated change.
- The revision number of the current file.

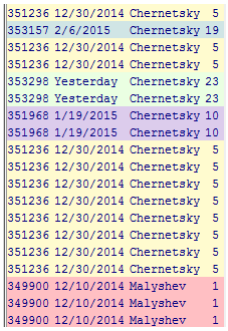
3. To hide annotations, right-click the annotations gutter, and choose Close Annotations.

Configuring the amount of information shown in the annotations pane

1. Enable annotations and right-click the annotations gutter.
2. Select View in the context menu and enable or disable the following options according to the type of information you want to be displayed:
   - Revision: select this option if you want to see the number of the changelist within which the annotated changes were checked in.
   - Date: select this option if you want to see the date when the annotated changes were checked in.
   - Author: select this option if you want to see the name of the user who checked in the annotated changes.
   - Commit number: select this option if you want to see the revision number of the current file.
   - Colors: use this control to toggle between the following highlighting modes:
     - Author: select this option if you want to highlight changes made by different authors with different colors:



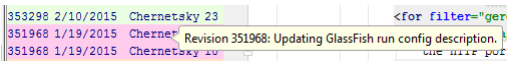
     - Order: select this option if you want annotation colors to indicate how recently a change was made. The entire file history is divided into several time periods containing an equal number of commits, and each time period is assigned its own color. The most recent changes are highlighted in green, while the oldest changes are highlighted in red:



- Hide: select this option if you do not want to use any color highlighting. In this case, all annotations will be displayed in gray.

- Names: use this control to select how user names will be presented. The following options are available:
  - Last name
  - First name
  - Full name

3. To view a commit message for an annotated change, hover the mouse cursor over an annotation. A tooltip will appear showing the commit message for the corresponding change:



Note that the amount of information displayed in the tooltip depends on the version control system you are using and is not affected by the annotation settings .

Annotating previous revisions

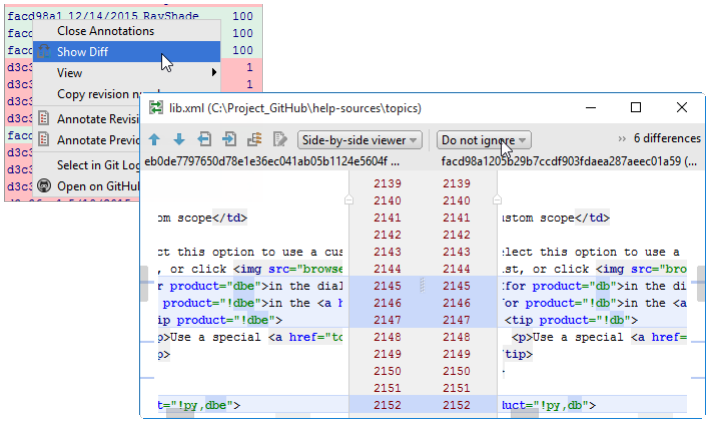
IntelliJ IDEA lets you annotate not only the current revision of the file, but also it's previous revisions. The following options are available:

- Annotate Revision: this option is useful if you want to check what a file looked like after a particular change was committed. To do this, right-click this change and select Annotate Revision from the context menu.
- Annotate Previous Revision: this option is useful if you find yourself in a situation when the last change in a particular line is meaningless, for example if all that was changed is code formatting. In this case, you can check what the previous revision of the file looked like. To do this, right-click a change and select Annotate Previous Revision from the context menu.

You can also annotate a particular file from the file history view . In the History tab, select the file version you want to review, right-click the corresponding line and select Annotate from the context menu.

Viewing differences between revisions

Position the cursor on an annotation, right-click it and select Show Diff from the context menu. IntelliJ IDEA opens the Differences viewer for files where you can view what changed between the annotated revision of the file and its previous revision:



Navigating to log

Position the cursor on an annotation, right-click it and select Select in Git log from the context menu. IntelliJ IDEA opens the Log Tab where you can view the specific revision.

The same result can be achieved by choosing the Copy revision number command from the context menu. Using this revision number, you can find the desired revision in the log.

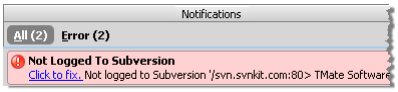
Accessing the Authentication to Server Dialog

For a number of reasons of various nature you may be not authenticated to the remote server while the authentication dialog box does not appear, as you might expect. Instead, IntelliJ IDEA displays the corresponding message in a pop-up window in the bottom-left corner of the editor.

Tip The authentication dialog appears immediately only when you attempt to access a URL address outside your working copies. This feature applies to Perforce, Subversion, and CVS.

To access the authentication dialog box

1. Click the Notifications Pending button on the Status bar.
2. In the Notifications pop-up window that opens, click the link Click to fix.



3. In the authentication dialog box that opens, specify your credentials.

VCS-Specific Procedures

In most cases, IntelliJ IDEA provides a unified approach to version control operations, as described in the previous sections. Nevertheless, there are certain VCS-specific features and peculiarities that the user should be aware of. Find helpful tips and notes in the following sections:

- Using ClearCase Integration
- Using CVS Integration
- Using Git Integration
- Using Mercurial Integration
- Using Perforce Integration
- Using Subversion Integration
- Using TFS Integration
- Using Visual SourceSafe Integration
- Using GitHub Integration

Using ClearCase Integration

With the ClearCase integration enabled, you can perform basic ClearCase operations from inside IntelliJ IDEA.

Note The information provided in the topics listed below assumes that you are familiar with the basics of ClearCase version control system.

In this section:

- Using ClearCase Integration
  - Prerequisites
  - ClearCase Support

Prerequisites

- ClearCase client is installed on your computer.
- IntelliJ IDEA comes bundled with ClearCase plugin. This plugin is turned on by default. If it is not, make sure that the plugin is enabled.
- ClearCase settings are defined in the ClearCase page of the Settings dialog.
- ClearCase integration is enabled for the current project root or directory.

Tip You can enable ClearCase integration for any directory. However, since it is not possible to configure a ClearCase view in IntelliJ IDEA, such directory should be already a ClearCase object, i.e. should exist within a ClearCase view.
- As of this writing, CCRC client is not supported.

ClearCase Support

- When ClearCase integration with IntelliJ IDEA is enabled, the ClearCase item appears on the VCS menu, and on the context menus of the Editor and Project views.
- The files in the folders under the ClearCase control are highlighted according to their status. See File Status Highlights for file status highlighting conventions.
- Modifications results are shown in the Version Control tool window .
- When using ClearCase integration, it is helpful to open the Version Control tool window. The Console tab displays the following data:
  - All commands generated based on the settings you specify through the IntelliJ IDEA user interface.
  - Information messages concerning the results of executing generated ClearCase commands.
  - Error messages.

Using CVS Integration

With the CVS integration enabled, you can perform basic CVS operations from inside IntelliJ IDEA.

Note The information provided in the topics listed below assumes that you are familiar with the basics of CVS version control system.

In this section:

- Using CVS Integration
  - Prerequisites
  - CVS Support
- Browsing CVS Repository
- Checking Out Files from CVS Repository
- Configuring CVS Roots
- Configuring Global CVS Settings
- Ignoring Files
- Importing a Local Directory to CVS Repository
- Resolving Commit Errors
- Updating Local Information in CVS
- Using CVS Watches
- Working Offline
- Working with Tags and Branches

Prerequisites

- IntelliJ IDEA comes bundled with the CVS plugin. This plugin is turned on by default. If it is not, make sure that the plugin is enabled.
- IntelliJ IDEA CVS integration does not require a standalone CVS client. All you need is an account in your CVS repository.
- CVS integration is enabled for the current project root or directory.

CVS Support

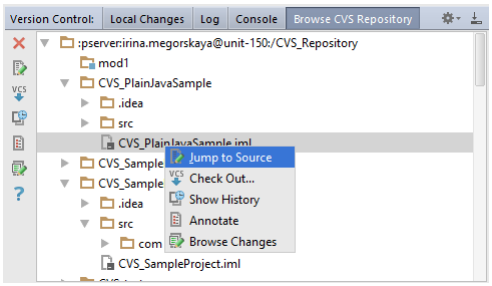
- When CVS integration with IntelliJ IDEA is enabled, the CVS item appears on the VCS menu, and on the context menus of the Editor and Project views.
- The files in the folders under the CVS control are highlighted according to their status. See File Status Highlights for file status highlighting conventions.
- Modifications results are shown in the Version Control tool window .
- When using CVS integration, it is helpful to open the Version Control tool window. The Console tab displays the following data:
  - All commands generated based on the settings you specify through the IntelliJ IDEA user interface.
  - Information messages concerning the results of executing generated CVS commands.
  - Error messages.

Browsing CVS Repository

You can browse any CVS repository and modify the structure of the currently open project, or a different one. Browsing contents of a repository is always available, even when CVS is not enabled in project. All you need is a valid user account.

To browse the CVS repository and modify its structure

1. Open a project. Then, choose VCS | Browse VCS Repository | Browse CVS Repository... on the main menu. The Select CVS Root Configuration dialog is opened.
2. Select a root from the list of configured CVS roots, or click Configure to specify a new one , and then click OK. The Browse CVS Repository tab opens in the CVS tool window at the bottom of the Editor.
3. Browse the desired CVS repository and perform jump to source, checkout, browse changes and annotate operations for files and folders.



Icons for the tree nodes denote their respective types. For example:

– means a CVS directory.

– denotes a CVS module.

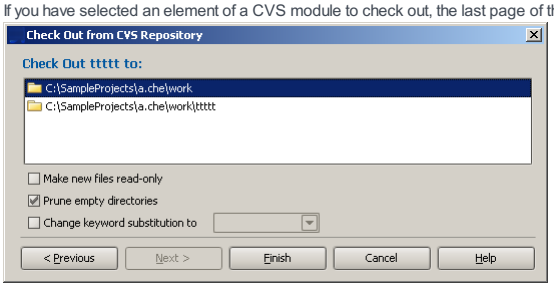
Icons appearing next to the files will denote the corresponding file types.

Checking Out Files from CVS Repository

Checking out action helps you obtain a writable copy of the repository, which you can edit as required. After making the necessary changes, you can publish results by committing, or checking in, to the repository. This section describes CVS-specific checkout procedure.

To check out files from a CVS repository

1. On the main menu, choose VCS | Checkout from Version Control.
2. On the submenu, choose CVS.
3. In the Checkout From CVS Repository dialog , select the desired CVS configuration, and click Next. If the necessary CVS configuration is not available, click Configure to create a new one .
4. Select the elements to check out, and click Next.
5. Specify the checkout destination directory where the local copy of the repository files will be created and click Next.
   If you are checking out sources for an existing project, the destination folder should be below a project content root .
6. If you have selected an element of a CVS module to check out, the last page of the wizard suggests you to add the module name to the local path:



7. Set up CVS checkout options, or accept defaults, and click Finish.
8. IntelliJ IDEA suggests to create a project based on the sources, checked out from version control.
   If you accept the suggestion, the New Project from Existing Code Wizard starts.

Configuring CVS Roots

CVS roots are global IntelliJ IDEA properties. Once configured, a CVS root is available regardless of which project is currently opened. This is helpful if you need to check out an entire project from CVS.

You can define multiple CVS root configurations for future use and edit them whenever necessary. Alternatively, you can configure CVS roots when checking out files or directories from or importing them to a CVS repository.

From this section you will learn how to:

- Configure a new CVS root
- Modify a CVS root
- Configure a CVS root based on an existing configuration
- Remove a CVS root configuration

To configure a CVS root, follow these general steps:

1. Open the CVS Roots dialog box.
2. Click the Add button on the toolbar.
3. Specify the CVS root string .
4. Specify the version to work with.
5. Specify additional connection settings .
6. Click the Test connection button to check that the specified settings ensure establishing successful connection to the CVS server.
7. Click OK to apply the specified settings and close the dialog box.

To modify an existing CVS root configuration

1. Open the CVS Roots dialog box.
2. Select the CVS root to modify.
3. Edit the settings as while configuring a new CVS root.
   Changes made here apply to the current CVS root configuration only.

To configure a new CVS root based on an existing configuration

1. Open the CVS Roots dialog box.
2. Select the CVS root to be used as the basis for a new configuration.
3. Click the Copy button on the toolbar or press `Ctrl+O`. The selected configuration is copied as a new CVS root.
4. Edit the newly created CVS root configuration, as required.

To remove a CVS root configuration

1. Open the CVS Roots dialog box.
2. Select the CVS root you want to remove and click the Remove button on the toolbar.

Accessing the CVS Roots Dialog Box

You can access the CVS Roots dialog box in several ways, depending on the general task you are currently performing.

To open the CVS Roots dialog box, do one of the following:

- When checking out files or directories from a CVS repository: on the main menu, choose VCS | Checkout from Version Control | CVS and in the dialog box that opens click the Configure button.
- When importing files or directories to a CVS repository: on the main menu, choose VCS | Import into CVS and in the dialog box that opens click the Configure button.
- When defining a CVS root to use in the future: open a file which is already under CVS control and choose VCS | CVS | Configure CVS Roots on the main menu.

Assembling a CVS Root String

CVS root strings are specified in the CVS Root text box of the CVS Roots dialog box.

The CVS Root string syntax is:

`[:method:][[user][:password]@]hostname[:[port]]/path/to/repository`.

You can obtain the valid string from your system administrator, or assemble the CVS root parameters into a correct string manually, or use the Edit by Field functionality, as described below.

To assemble the CVS Root parameters

1. In the CVS Roots dialog box, click the Edit by Field button next to the CVS Root text box. The Configure CVS Root Field by Field dialog box opens.
2. Choose the connection method and specify the user name, port, host, and repository. See the Configure CVS Root Field by Field dialog box reference for details.
3. Click OK. The dialog box closes and you return to the CVS Roots dialog box. The CVS Root text box displays the specified parameters assembled in a valid string.

Specifying a Version to Work With

By default, IntelliJ IDEA suggests you to check out the latest (HEAD) revision to work with. However, you can synchronize your local working copy with any previous revision from the repository.

Tip Make sure that the CVS root field is filled in with valid data.

To specify the version to work with

1. In the Use Version section of the CVS Roots dialog box, select the criterion to search for the desired version. The available options are:
   - By tag
   - By date

2. Click the Browse button next to the selected option and do one of the following depending on the selected search criterion:
   - If you have selected By tag, select the desired tag from the list of tags that opens. The list displays the tags obtained from the CVS server according to the specified CVS root string.
   - If you have selected By date, select the date and time in the calendar that opens.

   You can also type the desired revision number, tag, or date in the text box next to the selected option.

Specifying Additional Connection Settings

You can flexibly configure connection to the CVS server using additional settings. The set of relevant options depends on the connection method specified in the CVS root string. The available additional options for each of the supported connection methods are displayed in the lower part of the CVS Roots dialog box. See the CVS Roots dialog box reference for details.

Configuring Global CVS Settings

For a module associated with CVS you can specify global CVS settings, which includes character set, location of the password file, connection timeout etc.

To configure CVS global settings for a directory associated with CVS

1. On the main menu, choose VCS | CVS | Global Settings.
2. In the Global CVS Settings dialog box that opens, specify the global settings and click OK.

Tip Alternatively, you can define global CVS settings when configuring a CVS root. To invoke the Global CVS Settings dialog box, click the Global Settings button in the CVS Roots dialog box.

Ignoring Files

If you want your CVS integration to ignore certain unversioned files under CVS-associated directories, and skip them when performing update, import etc., add these files to the CVS ignore list, which is stored in the `.cvsignore` file.

The way CVS integration handles unversioned files depends on the general settings for file creation . If the new files, created with IntelliJ IDEA, are not put under version control automatically, you can add them to the ignore list using CVS command. You can put the `.cvsignore` file under CVS version control, and it will be recognized by all CVS clients.

To include an unversioned file to the ignore list

1. Select an unversioned file under a CVS-associated directory (by default, such files are brown).
2. On the main VCS menu, or on the context menu of the selection, choose CVS | Ignore. If `.cvsignore` file is not under CVS version control, proceed to the next step. If `.cvsignore` already exists and is under version control, IntelliJ IDEA adds the file in question to the ignore list silently.
3. If you want to put the ignore list under version control, in the Add File .cvsignore to CVS dialog box, click Add to CVS and optionally specify the desired keyword substitution .

Tip If you want to remove a file from the ignore list, you can only do it manually. Open the `.cvsignore` file for editing by pressing `F4` , and remove the lines for the files that should not be ignored by CVS.

Importing a Local Directory to CVS Repository

You can import an entire directory to your CVS repository, provided that you have the corresponding access rights. This action is helpful for putting a whole project under version control.

Import to the repository is always available, even when CVS is not enabled in project.

To import a directory into CVS repository

1. On the main menu, choose VCS | Import into CVS.
2. On the first page of the Import Into CVS wizard, select the target CVS root. If the desired target repository does not exist, you can create a new one. To do that, click Configure, and define the desired CVS root . Click Next.
3. On the second page of the wizard, select the target directory in the repository, and click Next.
4. On the third page of the wizard, select the local directory that will be imported into the CVS repository. Click Next.
5. In the Customize Keyword Substitution page, specify the keyword substitution rule for the files imported into the repository, and click Next.
6. Specify the required import settings . The fields of this page correspond to CVS command-line arguments for *import*, and additional options that define the checkout status of the imported directory.
7. Click Finish.

Resolving Commit Errors

In the section Checking in Files , you have learnt how to check in (commit) your changes to the repository. In this section you can find examples of CVS-specific error messages and suggestions on resolving conflicts.

If any error occurs when trying to commit, IntelliJ IDEA displays an error message. For example:

- If you have a modified file, which has been already changed on the server by someone else, since your last synchronization, you will get the following error:

```
Error: cvs server: Up-to-date check failed for 'source/com/...'
```

  In this case you would need to merge your local copy with the current revision in the repository. When the copies are merged, and all possible conflicts are resolved , so that the file is assigned the merged status, you can safely commit it to the repository.

- If you try to commit a file marked with a sticky tag, or sticky date, the CVS server will detect an attempt to *change the past*, and the error looks as follows:

```
Error: cvs server: sticky tag '1.1' for file 'source/com/impl/ManagerImpl.java& is not a branch
```

  To solve the problem, you need to update with resetting sticky data; in this case your changes will be merged with the most recent revision of the file. After resolving possible conflicts (by calling the Merge command) you will be able to commit the files.

Updating Local Information in CVS

From the Updating Local Information topic, you have learnt the general procedure. CVS integration provides a special Update File / Directory dialog box with the options that map directly to the corresponding CVS command-line options of the `update` command. Refer to the CVS documentation for details.

This section will consider the CVS-specific procedure and several of the options in terms of their presentation in IntelliJ IDEA.

To update local information

1. Select one or more files and/or directories in any navigation view (for example, Project tool window).
2. On the main menu, select VCS | Update Project or on the context menu of the selection choose CVS | Update File (Directory). The Update dialog for a project or file opens. In case of project update, select CVS tab.
3. Specify the following options:
   Branch Merging

   You can choose to merge your local file(s) with the counterpart(s) in one or two CVS branches. In the CVS command-line interface, this is the `-j` option.

   The option *Don't Merge* is selected by default, as merging across branches is not commonly needed. If you choose one of the other options, one or both of the text fields are enabled (depending on the choice of options). Clicking the Browse button … next to each field opens the Select Tag dialog box which lists all the branch tags maintained by the repository on the CVS server. Locate the branch you want to merge with, select it in the list, and click OK.

   Use Version

   You can optionally update your local system from some different revision. You can choose a revision by its tag or by its date. The Default option synchronizes with your file's current revision, as this is the most common synchronization. The other options for Use Version are:

   – By tag (-r):When updating a single file, you can choose the revision either by Revision or Tag. When you choose this option, the text field and corresponding ellipsis button are enabled. Enter a revision number or tag in the text field, or click the ellipsis button and select either a revision or a tag in the resulting dialog.

     When updating multiple files (selected individually, or when invoking update on a directory), you can only select the revision by Tag.

   – By date (-d): You can update from the revision of a specific date. This is possible whether you are invoking update on one file or multiple files or directories. When you choose this option, the date defaults to the current date. To specify a different date, click the ellipsis button and specify the desired date in the Choose Date dialog which appears.

   Reset sticky data (-A)

   If the last checkout or update of the selected file(s) was from some revision that was specified by tag or date, the tag/date information is *sticky* for the file(s). If you now want to update these sticky-tagged files from the *HEAD* revision, select this option in combination with selecting the Default option in *Use Version* so that the sticky information is removed.

   Change keyword substitution to

   If checked, this is converted to the -k CVS parameter. Refer to CVS Options: Default keyword substitution for text files for details.

   Do not show this dialog in the future

   Select this option if you want the update operation take place silently. For details see CVS Options.

   To have IntelliJ IDEA show this dialog box before update again:

   1. Open the Version Control - Confirmation page of the Settings dialog box.
   2. In the Display Option dialogs when these commands are invoked area, select the Update check box.

Using CVS Watches

CVS integration of IntelliJ IDEA helps coordinate the activities of team members, who concurrently work on the same files or directories.

CVS watches enable you to notify the users of a CVS repository whenever a file has been opened for editing, or committed. If you watch a file or directory for changes and commits, you are added to the list of watchers.

Usage the commands related to watches depends on configuration of the `$CVSROOT/CVSROOT/notify` file.

Edit and Unedit commands change read-only status of the files or directories under CVS. If the sources were checked out with the option `-c`, you can apply Edit command to make them writable. In this case you are added to the list of editors. When you are done with editing, use Unedit command to restore read-only status. So doing, you are removed from the list of editors. If watching is configured, the watchers will receive email notification about these events.

In fact, the option Use read-only flag for not edited files in the CVS provides the same functionality automatically. If this option is checked, Unedit always applies to the source files after commit.

Edit and Watch commands apply to all the files you have selected in the current view (including all the files in any selected directory), or to the current file in the editor if you invoked the command there.

This section describes how to:

- Access Edit and Watch commands
- Change read-only status of a file or directory
- View the other persons who edit the same file or directory
- Set watch on a certain event for a file or directory and thus add yourself to the list of wanchers
- Enable or disable watching
- View the other persons who watch the same file or directory

To access Edit and Watch commands

1. In one of the tool windows, select the desired files or directories, or open a file in the editor.
2. Do one of the following:
   - On the main menu, choose VCS | CVS | Edit and Watch
   - On the context menu, choose CVS | Edit and Watch

To get write access to a file or directory

1. Open Edit and Watch menu .
2. Choose Edit on the submenu. Edit Options dialog box is displayed.
3. If you want to gain exclusive write access, check the option Reserved edit (-c). Click OK.

To restore read-only status of a file or directory

1. Open Edit and Watch menu .
2. Choose Unedit on the submenu.

To view the other persons who edit the same file or directory

1. Open Edit and Watch menu .
2. Choose Show Editors on the submenu. This will display the list of all users who have run the Edit command on the same file or directory.

To set watch on a file or directory

1. Open Edit and Watch menu .
2. Choose Add Watch on the submenu.
3. In the dialog box that opens, select the type of action you would like to be notified about:
   - Edit: you will notified whenever Edit is applied to a watched file or directory.
   - Unedit: you will notified whenever Unedit is applied to a watched file or directory.
   - Commit: you will notified whenever Commit is applied to a watched file or directory.
   - All: you will notified whenever any of the above commands is applied to a watched file or directory.

To remove watch from a file or directory

1. Open Edit and Watch menu .
2. Choose Remove Watch on the submenu.
3. In the dialog box that opens, select the type of action for which you would like to skip notification (Edit, Unedit, Commit or All).

To suspend or resume watching

1. Open Edit and Watch menu .

2. Choose Watch Off or Watch On on the submenu.

To view the list of users who are watching the same files or directories

1. Open Edit and Watch menu .

2. Choose Show Watchers on the submenu.

Working Offline

Offline mode in CVS makes it possible to ignore network errors. When this mode is enabled, you will not receive any notifications about connection problems.

You can go to offline mode in two ways

- When a connection error occurs, IntelliJ IDEA informs you about that. With the first successful transaction, offline modes automatically turns off.
- Using the CVS menu command on the main Version Control menu, or a context menu: VCS | CVS | Work Offline.

Working with Tags and Branches

From within IntelliJ IDEA, you can create and delete CVS tags and branches. The names of the tags and branches must start with a letter, and contain only alphanumeric characters.

Branch and tag commands apply to all the files you have selected in the current view (including all the files in any selected directory), or to the current file in the editor if you invoked the command there.

This section describes how to:

- Access the tags and branches commands .
- Create a branch in the repository on the base of the current revision.
- Tag a revision in you local working directory .
- Delete a tag .

To access tags and branches commands

1. In one of the tool windows, select the desired files or directories, or open a file in the editor.
2. Do one of the following:
   - On the main menu, choose VCS | CVS
   - On the context menu, choose CVS
3. On the submenu, choose the appropriate command (Create Branch, Create Tag, or Delete Tag)

To create a branch

1. Invoke the Create Branch command.
2. In the Create Branch dialog box, specify the name of the new branch. To do that, type the name in the Branch name field, or click the Browse button and select the desired name from the list of existing CVS tags.
3. Optionally, specify the following:
   - Select the Override existing check box, if you want to move an existing tag to a new branch, as defined by the option `-F` of `rtag` CVS command.
   - Select the Switch to this branch check box to switch your local working copy to the branch specified in the Branch name field.

To create a new tag

1. Invoke the Create Tag command.
2. In the Create Tag dialog box, specify the new tag name. To do that, type the name in the Tag name field, or click the Browse button and select the desired name from the list of CVS tags.
3. Optionally, specify the following:
   - Select the Override existing check box, if you want an existing tag to point to the current revision, as defined by the option `-F` of `rtag` CVS command.
   - Select the Switch to this tag check box to switch your local working copy to the tag specified in the Tag name field.

To delete a tag

1. Invoke the Delete Tag command.
2. In the Delete Tag dialog box, specify the name of the tag you want to delete. To do that, type the name in the Tag name field, or click the Browse button , and select tag name from the list of the current tags in the repository.

Using Git Integration

With the Git integration enabled, you can perform basic Git operations from inside IntelliJ IDEA.

Note The information provided in the topics listed below assumes that you are familiar with the basics of Git version control system.

In this section:

- Using Git Integration
  - Prerequisites
  - Git Support
- Adding Files to a Local Git Repository
- Adding Tags
- Checking Git Project Status
- Committing Changes to a Local Git Repository
- Fetching Changes from a Remote Git Repository
- Handling Passwords for Git Remote Repositories
- Managing Branches
- Setting up a Local Git Repository
- Stashing and Unstashing Changes
- Pulling Changes from the Upstream (Git Pull)
- Pushing Changes to the Upstream (Git Push)
- Handling LF and CRLF Line Endings

Prerequisites

- Git is installed on your computer.
  It is strongly recommended that you use version 1.7.1.1 or higher.
- The location of the Git executable file is correctly specified on the Git page of the Settings dialog box.
- Git integration is enabled for the current project root or directory.
- If you are going to use a remote repository, create a Git hosting account first. You can access the remote repository through the username/password and keyboard interactive authentication methods supported by the Git integration or through a pair of `ssh` keys.
  Please note the following:
  1. `ssh` keys are generated outside IntelliJ IDEA. You can follow the instructions from https://help.github.com/articles/generating-ssh-keys/ or look for other guidelines.
  2. Store the `ssh` keys in the home_directory `\.ssh\` folder. The location of the home directory is defined through environmental variables :
     - `HOME` for Unix-like operating systems.
     - `userprofile` for the Microsoft Windows operating system.
  3. Make sure, the keys are stored in files with correct names:
     - `id_rsa` for the private key.
     - `id_rsa.pub` for the public key.
  4. IntelliJ IDEA supports a standard method of using multiple `ssh` keys, by means of creating `.ssh/config` file.

Git Support

- When Git integration with IntelliJ IDEA is enabled, the Git item appears on the VCS menu, and on the context menus of the Editor and Project views.
- The files in the folders under the Git control are highlighted according to their status. See File Status Highlights for file status highlighting conventions.
- Modifications results are shown in the Version Control tool window .
- When using Git integration, it is helpful to open the Version Control tool window. The Console tab displays the following data:
  - All commands generated based on the settings you specify through the IntelliJ IDEA user interface.
  - Information messages concerning the results of executing generated Git commands.
  - Error messages.

Adding Files to a Local Git Repository

On this page:

- Basics
- Adding all currently unversioned files to Git control
- Adding specific files to a local Git repository

Basics

After a Git repository for a project is initialized , you need to add the project data to it.

If you have specified Git as the version control system for your project in the Settings | Version Control page, IntelliJ IDEA suggests to put each newly created file under Git version control.

To make Git ignore some types of files, configure files to ignore .

Adding all currently unversioned files to Git control

1. Switch to the Version Control tool window and switch to the Local Changes tab. tool window.
2. Expand the Unversioned Files node and choose Add to VCS from the context menu or press `Ctrl+Alt+A`.

Adding specific files to a local Git repository

Do one of the following:

- Switch to the Version Control tool window and switch to the Local Changes tab. Expand the Unversioned Files node, and select the files to be added. From the context menu, choose Add to VCS, or press `Ctrl+Alt+A`.
- Switch to the Project tool window and select the files to be added. From the context menu, choose Git | Add or press `Ctrl+Alt+A`.

Adding Tags

On this page:

– Basics
– Assigning tags to commits
– Creating annotated tags

Basics

The Git integration supports tagging a particular commit or object to refer to it in the future. Basically, by adding a tag you create a branch that never moves.

You can also create annotated tags and have annotations displayed to facilitate viewing and navigating through tagged commits.

Assigning tags to commits

To assign a tag to a commit

1. On the main menu, choose VCS | Git | Tag Files. The Tag dialog box opens.
2. From the Git Root drop-down list, select the required local repository.
   The Current Branch read-only field shows the branch you are currently working with in the selected repository. The contents of the read-only field change as you change the selection in the Git Root drop-down list.
3. In the Tag Name text box, type the name of the new tag.

   Tip If you specify a name that already exists, IntelliJ IDEA displays an error message. To override the error and re-assign the tag, select the Force check box.
4. In the Commit text box, specify the commit or object which you want to tag:
   – To tag the latest commit in the branch (HEAD), leave the text box empty.
   – To tag a particular commit, specify its commit hash or use an expression, for example, of the following structure: `<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>`.
   Refer to the Git commit naming conventions for details.
5. To check that the specified commit exists and is the one you actually need, click the Validate button. The Paths affected in commit dialog box shows the files that were affected in the specified commit. View the information and click OK.
   If you specify a commit that does not exist, IntelliJ IDEA displays an error message.

Creating annotated tags

To create an annotated tag

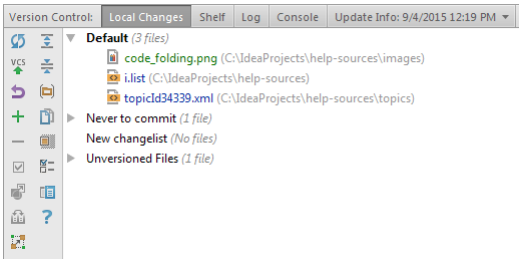
1. Create a tag.
2. In the Message text box, provide a description of the commit.

Checking Git Project Status

Besides indicating the current file status relative to the repository, IntelliJ IDEA integration with Git provides you with the accumulated view of the project files' statuses.

To view the differences between the current state of project files and the repository

1. Open the required project.
2. On the main menu, choose VCS | Refresh File Status.
3. Switch to the Version Control tool window and open the Local Changes tab.



The status of each file is indicated by the color in which the path to the file is displayed.

Committing Changes to a Local Git Repository

This topic describes how to commit changes to a local Git repository . For information on uploading changes to a remote repository, see Pushing Changes to the Upstream (Git Push) .

To commit changes to a local repository

1. Open the Version Control tool window by pressing `Alt+9` or choosing View | Tool Windows | Version Control.
2. In the Local Changes tab , select the files or folders you want to commit.
3. Open the Commit Changes dialog box by doing one of the following:
   - On the tool window toolbar, click the Commit Changes button .
   - On the main menu, choose VCS | Commit Changes.
   - On the main menu, choose VCS | Git | Commit File.
   - Press `Ctrl+K`.
4. The Commit Changes dialog lists all files that have changed since the last commit. Select the check boxes next to the files you want to commit now.
5. Add a comment in the Commit Message section and select the Before Commit options you want IntelliJ IDEA to perform on the files before committing them to the local repository.

   Tip To upload the committed changes to the remote repository immediately, select the Push Changes check box. The changes will be pushed to the pre-configured origin.
6. In the Git section, configure the following settings:
   - Use the Author drop-down list to specify the person who created the changes to be committed.

     Tip This may be necessary when you are committing changes made by another person.
   - If you want to replace already published commit, select the check box Amend commit, see Git Basics: Undoing Things for details.
7. Click Commit.

   Tip To save the changes in a text file, click the Create Patch button.

Fetching Changes from a Remote Git Repository

The Fetch operation involves downloading changes from a remote repository without applying them locally.

Tip Basically, the Fetch operation is intended for downloading changes from a remote repository that is different from the Origin .

To fetch changes from a remote repository

- On the main menu, choose VCS | Git | Fetch .

  IntelliJ IDEA retrieves the changes from the repository silently.

Handling Passwords for Git Remote Repositories

On this page:

- Basics
- Configuring the password policy
- Setting the master password
- Changing the master password
- Resetting the master password

Basics

Every time you interact with a remote Git repository, for example, during Pull, Update, or Push operations, you are requested to specify the private `ssh` key or passphrase to identify yourself. You may happen to use a number of remote repositories and accordingly need to remember a number of passwords.

IntelliJ IDEA provides the ability to configure a password policy according to which passwords are either never saved, or saved during one session and cleared upon the session end, or stored in a special passwords database.

The passwords database is under protection of a master password. Once specified , the master password can be changed or reset at any time. IntelliJ IDEA stores the history of all updates to the master password until it is reset.

Warning! When you reset the master password, the history of all the previously used master passwords is removed.

Configuring the password policy

To configure the password policy

1. Open the IDE Settings , then click Passwords.
2. On the Passwords page that opens, specify how you want IntelliJ IDEA to process passwords for Git remote repositories.
   Do one fo the following:
   - If you do not need IntelliJ IDEA to save passwords at all, select the Do not remember passwords option.
   - To have passwords stored in the memory during a session, select the Remember passwords until closing of the application option.
   - To have passwords stored in a passwords database, select the Remember on disk (protected with master password) option and specify the master password for the storage.

Setting the master password

To set the master password

1. On the Passwords page of the Settings dialog box, select the Remember on disk (protected with master password) option.
2. Click the Reset button.
3. In the Master Password dialog box that opens, specify the password to use. Type the desired password once again to confirm your setting.

Changing the master password

To change the master password

1. On the Passwords page of the Settings dialog box, click the Change Password button.
2. In the Change Master Password dialog box that opens, type the currently used master password and the password to change it to. Confirm the new password by typing it once again.

Resetting the master password

To reset the master password

1. Open the Reset Master Password dialog box. Do one of the following:
   - On the Passwords page of the Settings dialog box, click the Reset Master Password button.
   - In the Change Master Password dialog box, click the Reset Password button.
2. In the New Password text box, type the master password to replace the current one with.
3. In the Confirm Password text box, type the new master password once again to confirm your setting.

Managing Branches

IntelliJ IDEA supports processing of multiple Git branches using sophisticated checkout and merge strategies, and provides an interface for configuring operations on branches.

To manage branches, use the Branches node and the nested commands. This node is available on:

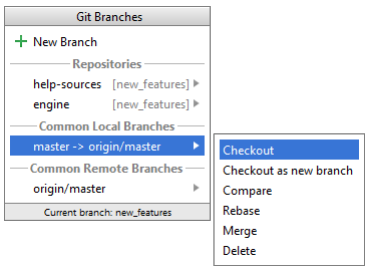
- the main VCS menu
- context menu of the editor
- VCS operations pop-up

In this part:

- Accessing Git Branches Popup Menu
- Creating a New Branch
- Checking Out (Switching Between) Branches
- Merging, Deleting, and Comparing Branches
- Rebasing Branches
- Resetting Head Commit
- Resetting Branch Head to a Selected Commit
- Applying Changes from a Specific Commit to Other Branches (Cherry Picking)
- Git Branches in Multirooted Projects

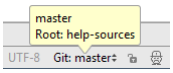
Accessing Git Branches Popup Menu

The Git Branches popup menu contains all commands that are required to manage Git branches, and can be invoked in a number of ways.



To open the Git Branches popup menu, do one of the following:

- From the main menu, choose VCS | Git | Branches.
- Right-click any file in the editor, and in the context menu choose Git | Repository | Branches.
- In the VCS operations popup menu, choose Branches.
- Click the Git widget in the Status bar:



Creating a New Branch

On this page:

- To create a new Git branch
- To check out a new Git branch from a local branch
- To check out a new local branch from a remote branch

To create a new Git branch

1. Invoke the Branches menu as described in Accessing Git Branches Popup Menu .
2. In the pop-up menu, choose New Branch.
3. In the Create new branch dialog box, specify the branch name. The branch with the specified name will be checked out (corresponds to `git checkout -b`).

To check out a new Git branch from a local branch

1. Invoke the Branches menu as described in Accessing Git Branches Popup Menu .
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Checkout as new branch from the submenu.
3. Specify the branch name in the Create new branch dialog that opens.

To check out a new local branch from a remote branch

1. Invoke the Branches menu as described in Accessing Git Branches Popup Menu .
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Checkout as new local branch from the submenu.
3. Specify the name of the new branch in the Checkout new branch from <branch name> dialog that opens.
   The branch with the specified name will be checked out and put under version control.

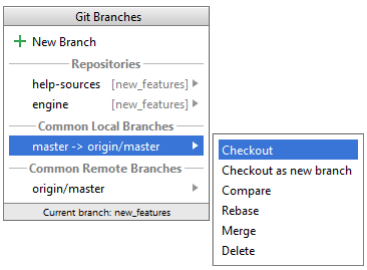
Checking out (Switching Between) Branches

On this page:

- Checking out an existing branch
- Checking which branch is currently checked out

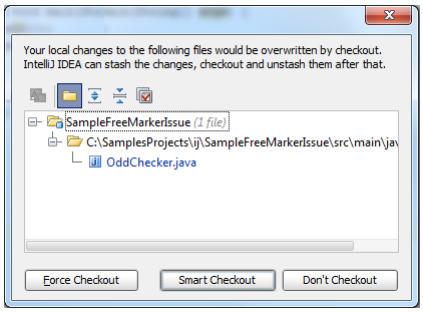
Checking out an existing branch

1. Invoke the Branches menu as described in Accessing Git Branches Popup Menu .
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Checkout from the submenu.



What happens next depends on whether there are conflicts:

- If your working tree is clean and your local changes do not conflict with the specified branch, this branch will be checked out (a notification will popup in the bottom-left corner of the IntelliJ IDEA window).
- If you ware working with a dirty tree, and your local changes will be overwritten by checkout, IntelliJ IDEA shows the files that prevent you from checking out the selected branch, and suggests to choose between the `force checkout` and `smart checkout` .



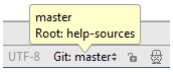
If you select `force checkout` , your local changes will be overwritten (equivalent to the `git checkout -f` command).

If you select `smart checkout` , IntelliJ IDEA will stash local changes, check out the selected branch, and then unstash the changes. If a conflict occurs during the unstash operation, you will be prompted to merge the changes.

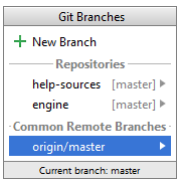
Checking which branch is currently checked out

If you want to check which branch is you are currently working in, do one of the following:

- Look ath the widget in the Status bar :



- Invoke the Branches menu as described in Accessing Git Branches Popup Menu . . Check the current branch name in the bottom line:



Merging, Deleting, and Comparing Branches

On this page:

- To merge a branch
- To delete a branch
- To compare branches

To merge a branch

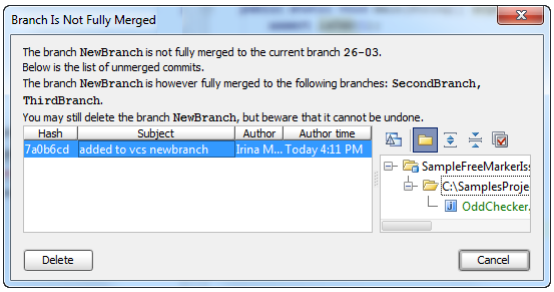
1. Invoke the Branches menu as described in Accessing Git Branches Popup Menu .
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Merge from the submenu. The selected branch will be merged into the branch that is currently checked out .
   If there are merge conflicts, you will be prompted to resolve them.

   If your local changes are going to be overwritten by merge, IntelliJ IDEA suggests to perform smart merge (similar to smart checkout .

To delete a branch

1. Invoke the Branches menu as described in Accessing Git Branches Popup Menu .
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Delete from the submenu.
   If there are no unmerged commits, the selected branch will be deleted.

   If there are commits that have not yet been merged into the selected branch, a dialog will open with a list of unmerged changes, and the list of branches where the selected branch has already been merged into. This information should help you decide if you still want to delete the selected branch.



   If you are sure you want to delete the selected branch, click Delete and confirm the operation.

To compare branches

1. Invoke the Branches menu as described in Accessing Git Branches Popup Menu .
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Compare from the submenu.
   IntelliJ IDEA compares the branch that is currently checked out with the selected branch.
3. In the dialog that opens, compare the differences in the following two tabs:
   - Log: this tab lists the commits that exist in the current branch, and are missing in the selected branch, and vice versa.
   - Diff: this tab shows the differences between files existing in both branches. Use the Show Diff command on the context menu of a file to explore the differences between branches.

Rebasing Branches

Git integration with IntelliJ IDEA supports the Rebase operation and provides an interface that ensures high flexibility in setting the rebase arguments.

The following functionality is supported:

- The basic use case which involves applying a branch on top of the current HEAD of the master after synchronization with the upstream.
- Rebasing a branch entirely or partially to a specific commit in any branch or tag.
- Running rebase on several local repositories simultaneously.
- Selecting a merge strategy to apply, with the possibility to use no merging strategy at all.
- Running rebase interactively with control over preserving/squashing merges.
- Resuming interrupted rebase after merge conflicts are resolved.
- Cancelling rebase.

To initiate a rebase operation

1. From the main menu, select VCS | Git | Rebase. The Rebase Branches dialog box opens.
2. From the Git Root drop-down list, select the relevant local repository.
3. From the Branch drop-down list, select the branch you want to rebase. By default, the current branch is selected. If you specify a different branch, it will be checked out.
4. Specify the new base and the commits you want to apply.
5. If necessary, choose a rebase strategy and click Rebase.

The `rebase` command is also available from the Git Branches popup in the submenu for the selected branch.

To resume an interrupted rebase operation

- From the main menu, choose VCS | Git | Continue Rebasing.
  Before resuming the rebase operation, view the log in the Version Control tool window.

If rebase was initiated and interrupted on two or more local repositories, the Continue Rebasing dialog box is displayed. Select the repository on which you want to resume the rebase operation from the Git Root drop-down list.

To cancel a rebase operation

- From the main menu, choose VCS | Git | Abort Rebasing

If rebase was initiated on two or more local repositories, the Abort Rebasing dialog box is displayed. Select the repository on which you want to cancel the rebase operation from the Git Root drop-down list.

Applying a Branch Entirely on Top of Master

Suppose you have `branch1` based on `master`. While you were working in `branch1`, some updates were committed to `master`.

The diagram below illustrates rebasing `branch1` so that it applies on top of the current HEAD of `master`.

By default, commits `1`, `2`, and `3` are applied one after another in the chronological order. To skip, edit, squash commits or change their order, run the rebase operation in the interactive mode.

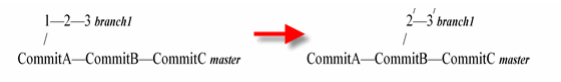
To apply a branch entirely on top of the current HEAD of the master

1. Initiate the rebase procedure.
2. Clear the Preserve Merges check box.
3. Clear the Interactive check box.
4. From the Onto drop-down list, select the master branch.
5. Clear the selection in the From drop-down list, if anything is selected.
6. From the Merge Strategy drop-down list, select Default.
7. Click the Rebase button. The rebase process starts. View the rebase log in the Version Control tool window, resolve conflicts that arise, and resume rebasing.

Rebasing a Branch to a Specific Commit

Suppose you have `branch1` based on `master`. While you were working in `branch1`, some updates were committed to `master`. For some reason, you do not want to rebase `branch1` to the current HEAD of `master`, but want to rebase it to commit `CommitB`. Moreover, you do not want to apply commit `1`, but want to start with commit `2`.

The diagram below illustrates the described rebase operation.



Transplanting a branch to a branch different from its base is also supported. Suppose your `branch2` is based on `branch1`, which, in its turn, is based on `master`.

The diagram below illustrates rebasing `branch2` to a specific commit in `master`.

To rebase a branch to a specific commit

1. Initiate the rebase procedure.
2. Configure the rebase mode:
   - To interactively handle the list of commits to apply, select the Interactive check box.
   - To have IntelliJ IDEA try to recreate merges instead of ignoring them, select the Preserve Merges check box.
     Note that this option is available only in the interactive mode.
3. Define the contents of the Onto and From drop-down lists. By default, tags and remote branches are not available from them. To expand the lists with tags or remote branches, select the Show Tags and Show Remote Branches check boxes respectively.
4. From the Onto drop-down list, select the required branch and specify the commit to move the base to. Type the desired commit hash or use an expression, for example, of the following structure:

   ```
   <branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>
   ```

   Refer to the Git commit naming conventions for details.

   Tip If you select a tag, no commit specification is required.
5. Click the Validate button and view view which files are affected in the specified commit in the Paths affected in commit... dialog box that opens.
6. From the From drop-down list, select the current branch and specify the commit, starting from which you want to apply commits on top of the new base. Type the desired commit hash or use an expression, for example, of the following structure:

   ```
   <branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>
   ```

   Refer to the Git commit naming conventions for details.

   Tip To apply all commits, leave the field empty.
7. Specify the merge procedure to be applied, when necessary:
   - To perform all merges manually, select the Do not use merging strategies check box.
   - To have IntelliJ IDEA apply one of the available merging methods, select the relevant option from the Merge Strategy drop-down list. See the Rebase Branches dialog box reference for description of available options.
8. Click the Rebase button and proceed depending on the chosen rebase mode:
   - If you have specified the interactive mode, configure the list of commits to apply in the Rebasing Commits dialog box, that opens.
   - If you have specified the automatic mode, view the rebase log in the Version Control tool window, resolve conflicts that arise, and resume rebasing

Interactive Rebase

Rebasing branches interactively provides you with the possibility to apply commits in the necessary order, squash or edit commits before applying, and skip commits that contain extraneous changes.

Interactive mode is available for rebasing branches entirely or partially on top of the HEAD or to any specific commit.

To rebase a branch in the interactive mode

1. Initiate the rebase procedure.
2. Select the Interactive check box.

   Tip To have IntelliJ IDEA try to recreate merges instead of ignoring them, select the Preserve Merges check box.

   Warning! Git does not support squashing commits when the Preserve merges option in enabled.
3. Specify the new base , the range of commits to apply, and the merge strategy .
4. Click OK. The Rebasing Commits dialog box opens displaying a list of all the commits within the specified range in the chronological order. For each commit its hash and comment are shown.

   Tip To view which files are affected in a commit, select the commit and click the View button.
5. Define the order of processing the commits by selecting the relevant lines and clicking the Move Up and Move Down buttons.
6. Use the Action drop-down list to define how each commit should be processed:
   - To apply a commit as is, select the Pick option.
   - To update a commit before applying, select the Edit option.
   - To ignore a commit, select the Skip option.
   - To combine a commit with the previous commit, select the Squash option.
     After you start rebasing, you will be asked to supply additional information on the squashed commits.

     If the affected commits have different authors, the squashed commit will be attributed to the author of the first commit.
7. Click the Rebase button. The rebase process starts. View the rebase log in the Version Control tool window, resolve conflicts that arise, and resume rebasing.

Resetting Head Commit

Suppose, you notice a small error in a recent commit or a set of commits and want to redo that part without showing the undo in the history. In this case resetting the latest commit in the current branch (HEAD) to an earlier commit is helpful. The Reset HEAD operation sets the current HEAD to the specified commit and optionally resets the index and working tree to match.

To reset the current HEAD to an earlier commit

1. On the main menu, choose VCS | Git | Reset HEAD. The Reset Head dialog box opens.

2. From the Git Root drop-down list, choose the required local repository.

3. Make sure the Current Branch read-only field shows the branch in which you actually want to reset the HEAD. If necessary, switch to the correct branch.

4. From the Reset Type drop-down list, select the desired reset strategy to apply:
   - To have the changed files preserved but not marked for commit, select the Mixed option. The index will be reset while the working tree will not and you will get a report of what has not been updated.

     Tip This is the default strategy.

   - To have only the HEAD pointer moved without updating the index and the working tree, select the Soft option. Your current state with any changes will remain different from the commit you are switching to.
   - To have both the working directory and the index changed to the specified commit, select the Hard option.

5. In the To Commit text box, specify the commit you want to reset the current HEAD to. Type its commit hash or use an expression, for example, of the following structure:
   `<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>`.

   Refer to the Git commit naming conventions for details.

6. To check that the specified commit exists and is the one you actually need, click the Validate button. The Paths affected in commit dialog box shows the files that were affected in the specified commit. View the information and click OK.
   If you specify a commit that does not exist, IntelliJ IDEA displays an error message.

Resetting Branch Head to a Selected Commit

IntelliJ IDEA allows you to reset the current Git branch head to a selected commit and update the working tree and the index.

To reset the current branch head to a selected commit

1. In the main menu navigate to View | Tool Windows | Changes.
2. In the Version Control tool window switch to the Log tab .
3. Select a commit that you want to reset the current branch head to, and right-click this commit. Select Reset Current Branch to Here from the popup menu.
4. In the Git Reset dialog that opens, select the mode for updating the working tree and the index. The following options are available:
   - Soft: The modified files will not be changed, and the differences will be staged for commit.
   - Mixed: The modified files will not be changed, and the differences will be unstaged. This mode is used by default and is identical to the `git reset` command.
   - Hard: The modified files will be reverted to the selected commit, and all uncommited changes will be lost.
   - Keep: The modified files will be reverted to the selected commit, but local changes will be preserved.

Applying Changes from a Specific Commit to Other Branches (Cherry Picking)

Suppose, you have two diverged branches, `master` and `hotfix`. One day, you decide to integrate changes from one specific commit in `master` to `hotfix`. With IntelliJ IDEA, you can cherry pick between branches right from the Version Control tool window.

To cherry pick changes between two branches

1. Switch to the target branch that the changes will be integrated to.
2. Open the Version Control tool window and switch to the Log tab.
3. From the Branch drop-down list on the toolbar, select the source branch containing the changes you want to cherry-pick.
   Note that the Commits pane lists all commits performed in the selected branch. To reduce the number of items in the list, you can filter the commits by the user or date.
   You can also click the Highlight non-picked commits button to grey out the commits that have already been applied to the current branch.
4. Select the required commit. Use the information in the Commit Details area if necessary.
5. Click the Cherry-pick button on the toolbar. IntelliJ IDEA will display the Commit Changes dialog with the automatically generated commit message.
6. If you want to review the changes or even modify the code before committing it to the target branch, you can do so in the difference viewer available from this dialog.
7. When done, click Commit to cherry-pick the selected changes.

Note that if you click Cancel in the Commit Changes dialog, a separate changelist will be created with the selected changes that you can see in the Local Changes tab . You can review these changes and commit them later if necessary.

Git Branches in Multirooted Projects

On this page:

- Basics
- Asynchronous branch control
- Synchronous branch control
- Rolling back a successful operation if it fails in some of the repositories

Basics

Sometimes you may need to have several local Git repositories within a single project. Such multirooted project can be set up in the following ways:

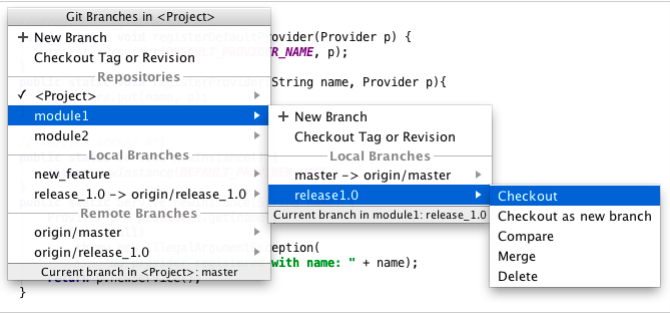
- The repositories are independent, they can be included in the project either as submodules or just as nested roots. One of these repositories can be intended for the project development itself while the others store the libraries used.
  Such repositories are controlled asynchronously , that is, commit, push, and other operations can be applied to them independently, just as it is done in projects with one Git repository.
- The repositories are mutually connected. Such repositories most often need to be controlled synchronously, that is, commit, push, and other operations should be applied to all these repositories at once. For such projects IntelliJ IDEA also allows you to control branches synchronously . This means that all branch operations on several repositories are performed simultaneously as if it were a single repository. In case an operation fails in at least one of the repositories, IntelliJ IDEA rolls it back in the repositories where the operation has passed successfully.

Asynchronous branch control

Asynchronous control is applied to the branches in a multirooted project when all local Git repositories in the project are independent. Commit, push, and other operations to the branches in the repositories are applied independently, just as it is done in projects with one Git repository.

To apply a Git operation to a branch asynchronously (in one of the repositories)

1. Open the Git Branches popup menu.
2. In the Git Branches popup menu that opens, choose the branch to apply the required action to. The popup menu shows a list of branches in the current repository and a list of all the repositories in the project.
   The current repository is detected in the following ways:
   - If you have explicitly selected a file or a folder in the Project tool window or elsewhere, IntelliJ IDEA treats the root under which the file is located as current repository.
   - If you used the VCS | Git | Git Branches command on the main menu, IntelliJ IDEA treats the root of the file currently opened in the editor as current repository.

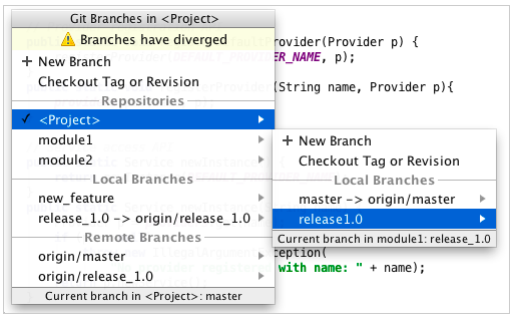


   Do one of the following:
   - To update a branch in the current repository, select the branch in the list.
   - To access another repository, select it under the Repositories group, and then select the relevant branch in the subordinate popup menu that opens.
3. Click the branch and choose the required action from the popup menu that opens.

Synchronous branch control

In a multirooted project when several mutually connected repositories, it is often helpful to apply synchronous branch control, which involves the following:

- All branch operations on several repositories are performed simultaneously as if it were a single repository. This applies to all actions including checkout , merge , and delete . If you invoke the compare action, IntelliJ IDEA shows a dialog box where you are asked to choose the repository to compare the branch with.
- The Git Branches popup lists only those local and remote branches that are common for all the repositories in the project.
- Creating a new branch from the popup will create it on all the repositories.
- You may still control branches individually for each repository by using the repository selector in the Git Branches popup menu. If you happen to checkout a branch only in one of the repositories, you'll get the warning that branches have diverged:



To disable synchronous branch control, do one of the following

When you invoke the Git Branches popup menu for the first time, IntelliJ IDEA examines the repositories registered in the project. If they all are on branches with same names, IntelliJ IDEA displays a popup menu that proposes to control branches synchronously.

- Click the Disable link in the information popup window.
- Open the Settings dialog box , and on the Git page under the Version Control node, clear the Control repositories synchronously check box.

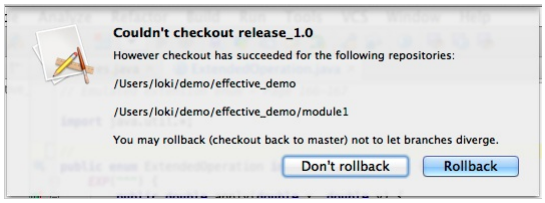
To enable synchronous branch control

- Open the Settings dialog box , and on the Git page under the Version Control node, select the Control repositories synchronously check box.

Rolling back a successful operation if it fails in some of the repositories

If you checkout a branch on several roots and the operation fails for some of the roots but has already succeeded for the others, the branches diverge. For example, the `master` branch is checked out in one repository while the `feature` branch is checked out in the other.

To prevent this, IntelliJ IDEA offers to rollback the successful checkout operation, so the previous checked out branch is checked out again.



Besides checkout, IntelliJ IDEA suggests to rollback merge and other branch operations.

Setting Up a Local Git Repository

Although Git provides high flexibility in arranging data and your work with repositories, the following scenarios are most commonly used for setting up a local Git repository:

- Clone an existing remote repository and create a new project with the downloaded data.
- Create a local repository which you can push to a remote location later, if necessary.

Warning! Git does not support external paths. So if you choose another directory, note that it must contain the tree where the project root resides.

To clone a remote Git repository

1. On the main menu, choose VCS | Checkout from Version Control | Git. The Clone Repository dialog box opens.
2. In the Git Repository URL text box, type the URL of the remote repository which you want to clone.
3. Click the Test button next to the Git Repository URL text box to check that connection to the remote repository can be established successfully.
4. In the Parent Directory text box, specify the directory where you want IntelliJ IDEA to create a folder for your local Git repository. Use ..., if necessary.
5. In the Directory Name text box, specify the name of the new folder into which the repository will be cloned. Click Clone.
6. Create a new project based on the cloned data by accepting the corresponding suggestion displayed by IntelliJ IDEA.

Git root mapping will be automatically set to the project root directory.

To create a local Git repository

1. Open the project you want to store in a repository.
2. On the main menu, choose VCS | Import into Version Control | Create Git Repository.
3. In the dialog that opens , specify the directory where you want to create a new Git repository.
4. Put the required files under Git version control. The files appear in the Local Changes tab of the Version Control tool window, under the Default changelist.

Tip By default, IntelliJ IDEA suggests the root of the current project.

- If you specify Git as the version control system for a directory in the Version Control dialog box, IntelliJ IDEA will suggest to put each new file in this directory under Git control.

Stashing and Unstashing Changes

On this page:

- Stashing and unstashing
  - Saving modifications to a new stash
  - Applying a stash
  - Removing stashes
- Stashing vs Shelving

Stashing and unstashing

Sometimes it may be necessary to revert your working copy to match the HEAD commit but you do not want to lose the work you have already done. This may happen if you learn that there are upstream changes that are possibly relevant to what you are doing or if you need to make some urgent fixes.

Stashing involves recording the difference between the HEAD commit and the current state of the working directory (stash). Changes to the index can be stashed as well.

Unstashing involves applying a stored stash to a branch.

You can apply a stash to an existing branch or create a new branch on its basis.

A stash can be applied as many times as you need to any branch you need, just switch to the required branch. Keep in mind that:

- Applying a stash after a series of commits results in conflicts that need resolving.
- A stash cannot be applied to a "dirty" working copy, that is a working copy to which changes have been made since the latest commit.

You can store stashes in a list as long as you need and remove them from the list when necessary.

Saving modifications to a new stash

To save modifications to a new stash

1. On the main menu, choose VCS | Git | Stash Changes. The Stash dialog box opens.
2. Select the relevant Git root and make sure that the correct branch is checked out.
3. In the Message text box, describe the changes to be stashed.

   Tip To stash the local changes and bring the changes staged in the index to your working tree for examination and testing, select the Keep Index check box.

Applying a stash

To apply a stash to the same branch

1. On the main menu, choose VCS | Git | Unstash Changes. The Unstash Changes dialog box opens.
2. Select the Git root where you want to apply a stash and make sure that the correct branch is checked out.
3. In the Stashes list, select the relevant stash.

   Tip Examine the stash descriptions and the names of the branches where specific stashes were created to find the stash you need.
4. Click the View button to open the Paths affected in commit dialog box and find out which files are affected in the selected stash.
5. Specify additional unstash options:
   - To have the selected stash removed from the list after it is applied, select the Pop stash check box.
   - To have the stashed index modifications applied, select the Reinstate Index check box.

   Warning! This operation may fail if you have conflicts. This happens because conflicts are stored in the index, where you can no longer apply the changes as they were originally.

To create a new branch on the basis of a stash

1. On the main menu, choose VCS | Git | Unstash Changes. The Unstash Changes dialog box opens.
2. Select the Git root where you want to apply a stash.
3. In the Stashes list, select the relevant stash.
4. In the As new branch text box, type the name of the new branch to be created.

Removing stashes

To remove one or several available stashes

1. On the main menu, choose VCS | Git | Unstash Changes. The Unstash Changes dialog box opens.
2. In the Stashes list, select the stashes to be removed and click the Drop button.

   Tip To remove all stashes from the list, click the Clear button.

Stashing vs Shelving

As you might have noticed, Stashing and Unstashing have much in common with Shelving and Unshelving respectively.

The only difference is in the way patches are generated and applied:

- Patches with stashed changes are generated by Git itself. To apply them later, you do not need IntelliJ IDEA.
- Patches with shelved changes are generated by IntelliJ IDEA. Normally, they are also applied through the IDE. Applying shelved changes outside IntelliJ IDEA is also possible but requires additional steps.

Pulling Changes from the Upstream (Git Pull)

Refreshing a local Git repository with the changes from the remote repository (Pull) involves retrieving changes (Fetch) and applying them to the local data (Merge). The Git integration with IntelliJ IDEA provides interface for specifying the mandatory Pull settings and for customizing the Pull procedure.

To pull changes from a remote repository

1. On the main menu, choose VCS | Git | Pull Changes. The Pull Changes dialog box opens.
2. Specify the required Git root and the URL address or alias of the source remote repository.
3. From the Branches to Merge list, select the remote branches you want to merge to the current local branch.
4. Specify the pull strategy and additional settings using the dialog box controls described in the Pull Changes dialog box reference.

Pushing Changes to the Upstream (Git Push)

The Git integration in IntelliJ IDEA supports uploading changes from the current branch to its tracked branch or to any other remote branch. If the upload (push) is rejected due to lack of synchronization between your local repository and the remote storage, you can either interactively choose the strategy to update the local branch or have IntelliJ IDEA update it silently.

Before uploading changes from your local Git repository to a remote repository make sure that:

- You have a Git hosting account.
- The target remote Git repository is accessible for your account.
- You have a pair of `ssh keys` to access the remote repository.

In this section:

- Pushing changes
- Updating a local branch interactively if push is rejected
- Configuring automatic update if push is rejected

To push changes

1. On the main menu, choose VCS | Git | Push . The Git Push dialog box opens listing all commits made in the current branch since the last `push` .
2. Select the commit to push. The right-hand pane shows the changes included in the selected commit. Use the toolbar buttons to examine the commit details.
3. Select which files you want to upload. To select all files, click the Select All toolbar button or press `Ctrl+A` .
4. Specify the target remote branch to push the changes to. If you want to upload the current branch to an alternative branch, rather than its tracked branch, click the branch name to edit it (note that completion is available for branch names).
   You can also click the Edit all targets link in the bottom right corner to edit all branch names simultaneously.
5. Click the Push button, when ready. IntelliJ IDEA starts the upload.
6. If `push` is rejected due to lack of synchronization, update the local branch in one of the following ways:
   - Choose the update strategy manually in the Push Rejected dialog box.
   - Have the local branch updated automatically .

To update a local branch interactively if push is rejected

When `push` is rejected due to lack of synchronization between the current local branch and its tracked remote branch, IntelliJ IDEA displays the Push Rejected dialog box, provided that the Auto-update if push ... rejected check box in the Git page of the Settings dialog box is cleared.

1. If your project uses several local Git repositories, specify the ones to be updated:
   - To have all the local repositories updated, no matter whether `push` has been rejected for all of them or not, select the Update non rejected repositories as well check box.
   - To have only affected repositories updated, clear the Update non rejected repositories as well check box.
2. In the Push Rejected dialog box, specify the way to update local repositories in the future:
   - To have this dialog box displayed upon every reject of `push` , clear the Remember the update method choice and silently update in future check box.
   - To have IntelliJ IDEA apply the update procedure invoked from this dialog box silently, select the Remember the update method choice and silently update in future check box.
     After you leave the dialog box, the Auto-update if push ... rejected check box in the Git page of the Settings dialog box is selected and the applied update method becomes default. To change this setting, clear the Auto-update if push ... rejected check box.
3. Invoke the update procedure rebase or merge by clicking the buttons Rebase or Merge respectively.

To have IntelliJ IDEA update a local branch automatically if push is rejected

Do one of the following:

- Open the Settings dialog box , click Git under the Version Control node, and in the in the Git page, that opens, select the Auto-update if push ... rejected check box.
  If you have not invoked any update operation from the Push Rejected dialog box before and you are enabling the check box initially, IntelliJ IDEA will synchronize the local branch silently by means of the merge operation.

  If you have previously invoked an update operation rebase or merge from the Push Rejected dialog box and selected the Remember the update method choice... check box, IntelliJ IDEA will apply your last choice.

  Accordingly, to change the "remembered" setting, clear the check box, access the Push Rejected dialog box, select the Auto-update if push ... rejected check box, whereupon invoke another update strategy.
- During an interactive update in the Push Rejected dialog box, select the Remember the update method choice and silently update in future check box, whereupon invoke the required update method `rebase` or `merge` by pressing the corresponding button. IntelliJ IDEA saves your choice and will apply it to update the conflicting local branch silently until you change the remembered setting .

Handling LF and CRLF Line Endings

On this page:

- Basics
- Enabling smart handling of LF and CRLF line separators
- Handling the problems with line separators during commit

Basics

Quite often people working in a team and using the same repository or upstream prefer different operating systems. This may result in problems with line endings, because Unix, Linux, and OS X use `LF` and Windows uses `CRLF` to denote the end of a line. IntelliJ IDEA shows the discrepancies in line endings in the Difference Viewer Dialog , so you can fix them manually.

To have Git solve such problems automatically, you need to set the `core.autocrlf` attribute to `true` on Windows and to `input` on Linux and OS X. For more details on the meaning of the `core.autocrlf` attribute, see the article Mind the End of Your Line or Dealing with Line Endings . You can change the configuration manually by running `git config --global core.autocrlf true` on Windows or `git config --global core.autocrlf input` on Linux and OS X. However, IntelliJ IDEA can analyze your configuration, warn you if you are about to commit `CRLF` into the repository, and offer to set the `core.autocrlf` setting to `true` or `input` depending on the operating system used.

Enabling smart handling of LF and CRLF line separators

To enable smart handling of LF and CRLF line separators

To enable smart handling of `LF` and `CRLF` line separators

1. Open the Settings dialog box .
2. Under the Version Control node, click Git.
3. On the Git page that opens, select the Warn if CRLF line separators are about to be committed checkbox.

Handling the problems with line separators during commit

To handle problems with the line separators during commit

1. Enable smart handling of LF and CRLF line separators . After that, IntelliJ IDEA will show the Line Separators Warning Dialog every time you attempt to commit a file with `CRLF` separators, unless you have set any related git attributes on the affected file. In the latter case, IntelliJ IDEA supposes that you clearly understand what you are doing and excludes the file from analysis.
2. When the Line Separators Warning Dialog is displayed, do one of the following:
   - To ignore the warning and commit the file with `CRLF` separators, click the Commit As Is button.
   - To have the `core.autocrlf` attribute set to `true` or `input` depending on the operating system used before commit, click the Fix and Commit button. As a result, all the `CRLF` separators will be replaced with `LF` separators and committed into the repository. Note that the reverse operation will not be performed when you download the files into your working directory, that is, no `CRLF` will appear in place of `LF` .
   - To stop the commit procedure, click Cancel.
3. To suppress showing the dialog box in the future, select the Don's warn again check box.

Using Mercurial Integration

With the Mercurial integration enabled, you can perform basic Mercurial operations from inside IntelliJ IDEA.

Note The information provided in the topics listed below assumes that you are familiar with the basics of Mercurial version control system.

In this section:

- Using Mercurial Integration
  - Prerequisites
  - Mercurial Support
- Adding Files to a Local Mercurial Repository
- Setting up a Local Mercurial Repository
- Managing Mercurial Branches and Bookmarks
- Switching Between Working Directories
- Pulling Changes from the Upstream (Pull)
- Pushing Changes to the Upstream (Push)
- Tagging Changesets

Prerequisites

- Mercurial is installed on your computer.
- The location of the Mercurial executable file `hg.exe` is correctly specified on the Mercurial page of the Settings dialog box.
  If you followed the standard installation procedure, the default location is `/opt/local/bin` or `/usr/local/bin` for Linux and OS X and `/Program Files/TortoiseHG` for Windows.

  It is recommended that you add the path to the Mercurial executable file to the `PATH` variable. In this case, you can specify only the executable name, the full path to the executable location is not required. For more information about environment variables, see Path Variables .
- Mercurial integration is enabled for the current project root or directory.

If you want to use a remote repository, create a Mercurial hosting account first. You can access the remote repository through a pair of `ssh keys` or apply the username/password and keyboard interactive authentication methods supported by the Mercurial integration.

Mercurial Support

- When Mercurial integration with IntelliJ IDEA is enabled, the Mercurial item appears on the VCS menu, and on the context menus of the Editor and Project views.
- The files in the folders under the Mercurial control are highlighted according to their status. See File Status Highlights for file status highlighting conventions.
- Modifications results are shown in the Version Control tool window .
- When using Mercurial integration, it is helpful to open the Version Control tool window. The Console tab displays the following data:
  - All commands generated based on the settings you specify through the IntelliJ IDEA user interface.
  - Information messages concerning the results of executing generated Mercurial commands.
  - Error messages.

Adding Files To a Local Mercurial Repository

After a Mercurial repository for a project is initialized , you need to add the project data to it:

- If you have specified Mercurial as the version control system for your project in the Settings dialog box, IntelliJ IDEA suggests to put each new file under Mercurial control during the file creation.

  To have Mercurial ignore some types of files, configure files to ignore .
- You can add all unversioned files to Mercurial control or select files to add .

To add all currently unversioned files to Mercurial control

1. Switch to the Version Control tool window.
2. In the Local Changhes tab, navigate to the Unversioned Files node and choose Add to VCS from the context menu.

To add specific file(s) to a local Mercurial repository, do one of the following:

- Switch to the Version Control tool window, expand the Unversioned Files node, and select the files to be added. From the context menu, choose Add to VCS.
- Switch to the Project tool window and select the files to be added. From the context menu, choose Mercurial | Add to VCS.

Setting Up a Local Mercurial Repository

Although Mercurial provides high flexibility in arranging data and your work with repositories, the following scenarios are most commonly used for setting up a local Mercurial repository:

- Clone an existing remote repository and create a new project with the downloaded data.
- Create a local repository which you can push to a remote location later, if necessary.

To clone a remote Mercurial repository

1. On the main menu, choose VCS | Checkout from Version Control | Mercurial. The Clone Mercurial Repository dialog box opens.
2. In the Mercurial Repository URL text box, type the URL of the remote repository which you want to clone.
3. Click the Test Repository button next to the Mercurial Repository URL text box to check that connection to the remote repository can be established successfully.
4. In the Parent Directory text box, specify the directory where you want IntelliJ IDEA to create a folder for your local Mercurial repository. Use the Browse button … button, if necessary.
5. In the Directory Name text box, specify the name of the new folder into which the repository will be cloned. Click Clone.
6. Create a new project based on the cloned data by accepting the corresponding suggestion displayed by IntelliJ IDEA.

To create a local Mercurial repository

1. Open the project you want to store in a repository.
2. On the main menu, choose VCS | Import into Version Control | Create Mercurial Repository. The Create Mercurial Repository dialog box opens.
3. Specify the location of the new repository.
   - To have the repository created in the project root, choose the Create repository for the whole project option. IntelliJ IDEA will create the `.hg` directory in the project root folder.
     This option is selected by default.
   - To have a new repository created in another location, choose the Select where to create repository option and specify the path to the repository location in the text box below. Type the path manually or click the Browse button … and choose the relevant folder in the Select directory for hg init dialog box that opens.

     Warning! Mercurial does not support external paths. So if you choose another directory, note that it must contain the tree where the project root resides.

     If you choose a directory which is already under Mercurial control, IntelliJ IDEA opens the Directory Is Under hg dialog box, where you can choose to create a repository in the specified location or to stay in the parent repository.
4. Put the required files under Mercurial version control. The files appear in the Version Control tool window under the Default node.
   Note that if you specify Mercurial as the version control system for a directory in the Version Control dialog box, IntelliJ IDEA will suggest to put each new file in this directory under Mercurial control.

Managing Mercurial Branches and Bookmarks

With IntelliJ IDEA, you can use both named branches and light-weight branches (bookmarks) . IntelliJ IDEA provides interface for creating, merging, and switching between branches and bookmarks, see Switching Between Working Directories . You can also run commands in the embedded Terminal, see Working with Embedded Local Terminal .

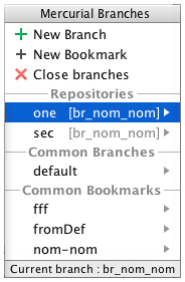
On this page:

– Opening the Branches pop-up list
– Creating a named branch
– Creating a bookmark
– Closing a branch
– Merging named branches and bookmarks
– Merging a named branch or bookmark with another named branch or bookmark
– Merging a named branch or bookmark with a changeset

For information about switching between branches and bookmarks, see Switching Between Working Directories .

Opening the Branches pop-up list

Most of the operations with branches and bookmarks are invoked from the Branches pop-up list.



The list shows all the Mercurial repositories under the project root, all the named branches, and all the bookmarks in them. The current repository and the current bookmark are marked with a tick. The name of the current named branch is shown in the dedicated hg area on the Status bar:

To open the Branches pop-up list, do one of the following:

– On the Status bar, click the name of the current named branch in the dedicated hg area.

– On the main menu, choose VCS | Mercurial | Branches.

– On the context menu of the Editor or Version Control tool window, choose Mercurial | Branches.

Creating a named branch

1. In the Branches pop-up list, click New Branch.
2. In the Create New Branch dialog box that opens, specify the name of the new branch.

The new branch immediately becomes active and its name is shown on the Status bar in the hg area.

Creating a bookmark

1. In the Branches pop-up list, click New Bookmark.
2. In the New Bookmark dialog box that opens, specify the name of the bookmark to be created.
3. Specify, whether you want to switch to the new bookmark immediately or not.
   – To activate the new bookmark and thus enable tracking and updating the light-weight branch the bookmarks identifies, leave the Inactive check box cleared. The new bookmark immediately becomes active and its name is marked with a tick in the Branches pop-up list.
   – To have an inactive bookmark created, that is, to remain in the current light-weight branch (bookmark) or named branch and switch to the new bookmark later, select the Inactive check box.

Closing a branch

According to Mercurial workflows , when you are done with a feature development and do not expect any further changes, you close the corresponding branch. A closed branch is not displayed among active branches, in the Log view , etc. To close a branch, do the following:

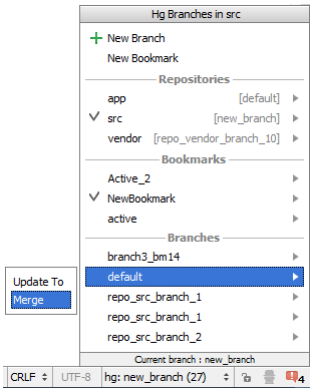
1. In the Branches popup, click Close branch. The Commit changes dialog will be displayed.
2. Click Commit and Close. All changes will be committed and the current branch will be closed.

Note that if you have several repositories listed in the Repositories section, the corresponding menu option will toggle to Close branches and the `close` operation will be applied to all of them.

Merging named branches and bookmarks

You can merge a named branch or a bookmark with another named branch, another bookmark, or a specific changeset identified by a tag or a revision number.

- Merging a named branch or a bookmark with another named branch or bookmark means merging with its head.
  Merging with named branches and bookmarks can be invoked through the menu item VCS | Mercurial | Merge, which opens the Mercurial-specific Merge dialog box of from the Branches pop-up list.



- Merging a named branch or a bookmark with a changeset means merging the branch head with the specified changeset. A changeset can be identified either by a revision number or a tag, see http://mercurial.selenic.com/wiki/Tag?highlight=%28\bCategoryGlossary\b%29 .
  Merging a named branch or a bookmark with a specific changeset can be invoked only through VCS | Mercurial | Merge.

For definitions and Mercurial-specific details regarding the merge operation itself, see http://www.selenic.com/hg/help/merge .

By default, Mercurial requires that before merge the current working directory should be clean, that is, it should not contain any uncommitted changes. Otherwise the merge operation fails and IntelliJ IDEA shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg merge <target branch, bookmark, or changeset> -C` to discard the uncommitted changes.

If your current working copy is not clean, you can either commit the changes or shelve them as described in Shelving and Unshelving Changes .

Merging a named branch or bookmark with another named branch or bookmark

Merging a named branch or a bookmark with another named branch or bookmark means merging with its head.

1. Make sure, your current working directory is clean, that is, it does not contain any uncommitted changes. Commit or shelve the changes, if any.

2. Invoke merge by doing one of the following:
   - In the Branches pop-up list, click the name of the branch or bookmark to merge with, then choose Merge on the pop-up menu:
   - Choose VCS | Mercurial | Merge on the main menu or Mercurial | Merge on the context menu of the Editor.
     In the Merge dialog box that opens:
     1. Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.
     2. Choose the Branch or Bookmark option and choose the named branch or bookmark to merge the current working directory with.

3. Resolve conflicts. As soon as a conflict takes place, the Files Merged with Conflicts dialog box opens with a list of conflicting files. Use the controls of the dialog box to resolve the problems:
   - To have the version of the current working directory preserved, click Accept Yours.
   - To have the version of the branch you are merging with preserved, click Accept Theirs.
   - To resolve the conflicts manually, click Merge and use the Conflict Resolution Tool, as described in Resolving Conflicts .

   If no conflicts arise during merge, the operation passes silently and the merge log is shown in the Version Control tool window.

Merging a named branch or bookmark with a changeset

Merging a named branch or a bookmark with a changeset means merging the branch head with the specified changeset. A changeset can be identified either by a revision number or a tag, see http://mercurial.selenic.com/wiki/Tag?highlight=%28\bCategoryGlossary\b%29 .

1. Make sure, your current working directory is clean, that is, it does not contain any uncommitted changes. Commit or shelve the changes, if any.

2. Choose VCS | Mercurial | Merge on the main menu or Mercurial | Merge on the context menu of the Editor.

3. In the Merge dialog box that opens:
   1. Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.
   2. Choose the Tag or Revision option and choose the tag or specify the hash or revision number to merge the current working directory with. To copy a hash, open the Log tab of the Version Control tool window, select the relevant branch and revision, and then choose Copy Hash on the context menu of the selection.

4. Resolve conflicts. As soon as a conflict takes place, the Files Merged with Conflicts dialog box opens with a list of conflicting files. Use the controls of the dialog box to resolve the problems:
   - To have the version of the current working directory preserved, click Accept Yours.
   - To have the version of the branch you are merging with preserved, click Accept Theirs.
   - To resolve the conflicts manually, click Merge and use the Conflict Resolution Tool, as described in Resolving Conflicts .

   If no conflicts arise during merge, the operation passes silently and the merge log is shown in the Version Control tool window.

Switching Between Working Directories

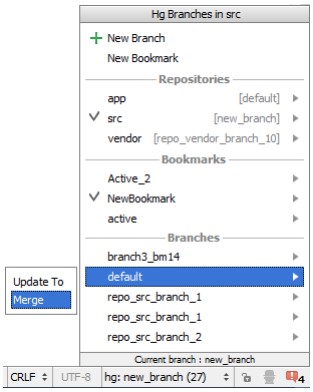
The Mercurial integration with IntelliJ IDEA provides the possibility to switch update the repository's working directory to the specified changeset or a specific line of development. Changesets can be identified by their hashes or by previously assigned tag identifiers.

On this page:

- Opening the Branches pop-up list
- Switching to another named branch or bookmark
- Switching to another changeset

You can update a named branch or a bookmark to another named branch, another bookmark, or a specific changeset identified by a tag or a revision number.

- Updating a named branch or a bookmark to another named branch or bookmark means updating to its head.
  Updating to named branches and bookmarks can be invoked through the menu item VCS | Mercurial | Update to, which opens the Mercurial-specific Switch Working Directory dialog box of from the Branches pop-up list.



- Updating a named branch or a bookmark to a changeset means updating the branch head to the specified changeset. A changeset can be identified either by a revision number or a tag, see http://mercurial.selenic.com/wiki/Tag?highlight=%28\bCategoryGlossary\b%29.
  Updating a named branch or a bookmark to a specific changeset can be invoked only through VCS | Mercurial | Update to.

By default, Mercurial requires that before update the current working directory should be clean, that is, it should not contain any uncommitted changes. Otherwise the update operation fails and IntelliJ IDEA shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg update <target branch, bookmark, or changeset> -C` to discard the uncommitted changes.

If your current working copy is not clean, you can either commit the changes or shelve them as described in Shelving and Unshelving Changes. IntelliJ IDEA provides the possibility to discard any uncommitted changes when the update operation is already invoked. This option is available only in the Mercurial-specific Switch Working Directory dialog box.

Opening the Branches pop-up list

To open the Branches pop-up list, do one of the following:

- On the Status bar, click the name of the current named branch in the dedicated hg area.

- On the main menu, choose VCS | Mercurial | Branches.
- On the context menu of the Editor or Version Control tool window, choose Mercurial | Branches.

Switching to another named branch or bookmark

Updating a named branch or a bookmark to another named branch or bookmark means updating to its head.

1. Make sure, your current working directory is clean, that is, it does not contain any uncommitted changes. Commit or shelve the changes, if any.
   If you invoke update through the Switch Working Directory dialog box, you can also prevent conflicts by having any uncommitted changes discarded.
2. Invoke update by doing one of the following:
   - In the Branches pop-up list, click the name of the branch or bookmark to update to, then choose Update to on the pop-up menu:
   - Choose VCS | Mercurial | Update to on the main menu or Mercurial | Update to on the context menu of the Editor.
     In the Switch Working Directory dialog box that opens:
     1. Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.
     2. Choose the Branch or Bookmark option and choose the named branch or bookmark to update the current working directory to.
     3. To prevent failures during update if the current working directory is not clean, select the Overwrite locally modified files (no backup) check box. The uncommitted changes will be discarded.

Switching to another changeset

1. Make sure, your current working directory is clean, that is, it does not contain any uncommitted changes. Commit or shelve the changes, if any.
   If you invoke update through the Switch Working Directory dialog box, you can also prevent conflicts by having any uncommitted changes discarded.
2. Invoke update by doing one of the following:
   - In the Branches pop-up list, click the name of the branch or bookmark to update to, then choose Update to on the pop-up menu:
   - Choose VCS | Mercurial | Update to on the main menu or Mercurial | Update to on the context menu of the Editor.
     In the Switch Working Directory dialog box that opens:
     Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.

1. Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.
2. Choose the Branch or Bookmark option and choose the named branch or bookmark to update the current working directory to.
3. To prevent failures during update if the current working directory is not clean, select the Overwrite locally modified files (no backup) check box. The uncommitted changes will be discarded.

3. Resolve conflicts. As soon as a conflict takes place, the Files Merged with Conflicts dialog box opens with a list of conflicting files. Use the controls of the dialog box to resolve the problems:
   - To have the version of the current working directory preserved, click Accept Yours.
   - To have the version of the branch you are merging with preserved, click Accept Theirs.
   - To resolve the conflicts manually, click Merge and use the Conflict Resolution Tool, as described in Resolving Conflicts .

   If no conflicts arise during update, the operation passes silently and the update log is shown in the Version Control tool window.

Pulling Changes from the Upstream (Pull)

Refreshing a local Mercurial repository with the changes from the remote repository (Pull) involves retrieving changes and applying them to the local data . The Mercurial integration with IntelliJ IDEA provides interface for specifying the mandatory Pull settings and for customizing the Pull procedure.

To pull changes from a remote repository

1. On the main menu, choose VCS | Mercurial | Pull Changesets. The Pull dialog box opens.

2. Specify the required URL address of the source remote repository.

Pushing Changes to the Upstream (Push)

Before uploading (Pushing) changes from your local Mercurial repository to a remote repository make sure that:

- You have a Mercurial hosting account.
- The target remote Mercurial repository is accessible for your account.
- You have a pair of `ssh keys` to access the remote repository.

To push changes from a local Mercurial repository

1. On the main menu, choose VCS | Mercurial | Push Changeset. The Push dialog box opens.
2. In the Destination Repository URL text box, specify the location of the remote repository to push.
3. Specify the changes to upload:
   - To have a branch uploaded entirely, select the Branch check box and choose the relevant branch from the drop-down list.
   - To have a specific changeset uploaded, select the Revision check box and specify the desired revision number in the text box. By default, the latest revision (tip) is uploaded. Learn more about changesets and tips at http://mercurial.selenic.com/wiki/UnderstandingMercurial .
4. To push a locally created branch or to avoid updating before upload, select the Force Push check box.

Tagging Changesets

IntelliJ IDEA supports both local and global tags. Local tags are stored in the file `.hg/localtags` in the repository, global tags are stored in the file `.hgtags`.

Currently tagging specific changesets is supported only in the command line mode in the embedded Terminal. To launch the Terminal, hover your mouse pointer over ▣ in the lower left corner of the IDE, then choose Terminal from the menu (see Working with Embedded Local Terminal for details). After commit, tags appear in the Log tab of the Version Control tool window.

For more information about Mercurial tags, see Mercurial Documentation: Tag.

Tagging a repository

IntelliJ IDEA provides UI for tagging the current repository which means assigning a tag to its tip. The created tag is global, is stored in the file `.hgtags`. After commit, the tag appears in the Log tab of the Version Control tool window.

1. Open the Tag dialog box by doing one of the following:
   - On the main menu, choose VCS | Mercurial | Tag Repository.
   - On the context menu of the Editor, choose Mercurial | Tag Repository.
2. In the Tag dialog box that opens, specify the tag name. The name must be unique.

Using Perforce Integration

With the Perforce integration enabled, you can perform basic Perforce operations from inside IntelliJ IDEA.

Note The information provided in the topics listed below assumes that you are familiar with the basics of Perforce version control system.

In this section:

- Using Perforce Integration
  - Prerequisites
  - Preliminary steps
  - Perforce Support
- Enabling and Configuring Perforce Integration
- Handling Modified Without Checkout Files
- Integrating Perforce Files
- Resolving Conflicts with Perforce Integration
- Showing Revision Graph and Time-Lapse View
- Using Multiple Perforce Depots with P4CONFIG
- Working Offline
- Checking Perforce Project Status
- Attaching and Detaching Perforce Jobs to Changelists

Prerequisites

- A Perforce client is installed on your computer.
- You have an account with the Perforce depot.

To start using Perforce integration, perform the following preliminary steps

1. Create a client spec using your Perforce client.
2. Get source files from the Perforce depot using your Perforce client.
3. As soon as the local working copy is on your computer, associate your local directory with Perforce .
   After that you will be able to open source files for edit, and perform the usual Perforce-related tasks using IntelliJ IDEA.

Perforce Support

- When Perforce integration with IntelliJ IDEA is enabled, the Perforce item appears on the VCS menu, and on the context menus of the Editor and Project views.
- The files in the folders under the Perforce control are highlighted according to their status. See File Status Highlights for file status highlighting conventions.
- Modifications results are shown in the Version Control tool window .
- When using Perforce integration, it is helpful to open the Version Control tool window. The Console tab displays the following data:
  - All commands generated based on the settings you specify through the IntelliJ IDEA user interface.
  - Information messages concerning the results of executing generated Perforce commands.
  - Error messages.

Enabling and Configuring Perforce Integration

The Perforce Integration is disabled by default. If you want to perform Perforce-related operations right from IntelliJ IDEA, enable the integration at the IDE level and associate the project root or specific directories with Perforce. The general procedure is described in the section Enabling Version Control .

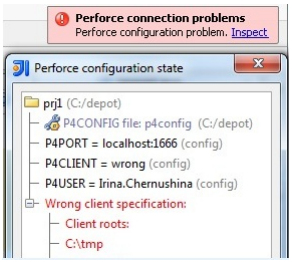
In this section:

– Enabling Perforce integration for a project or directory
– Configuring Peforce integration settings

To enable Perforce integration for a project or directory

1. Open the Settings dialog box , and click Version Control.
2. In the Version Control page, that opens, point to the desired root.
   This can be <Project Root> or a content root .
3. From the VCS drop-down list, choose Perforce.

If you specify a wrong client workspace, and your project roots do not match with the workspace roots, IntelliJ IDEA displays a pop-up window with a warning.



Click Inspect to view and fix the discrepancies.

To configure Perforce integration

1. Open the Settings dialog box , and then open the Perforce page under the Version Control node.
2. To establish live connection to the Perforce server, select the Perforce is online check box.
3. Specify which credentials you want to use for connecting to the Perforce server.
   – To use the connection settings from your `P4CONFIG` files, choose the Use P4CONFIG or default connection option.
   If you are using `P4CONFIG` files for configuration, IntelliJ IDEA shows what config files it has found and what other default settings are used. This way you can be sure that your `P4CONFIG` files are detected and taken into account.
   – To configure connection manually, choose the Use connection parameters option and specify the following settings in the corresponding text boxes:
      1. The Port the Perforce client will listen to.
      2. The Client name.
      3. Your User name and Password to authenticate to the server.
4. To use ticket-based authentication, select the Login authentication check box. Otherwise, password-based authentication will be used. In either cases IntelliJ IDEA uses the login name and password specified in the dialog box or stored in the `P4CONFIG` files.
5. To attempt to log on the Perforce server without authentication, select the Try to login silently check box.
6. To have IntelliJ IDEA create a `P4.output` file and store the output of Perforce commands in it, select the Dump Perforce Commands to: check box.
7. Specify the path to the Perforce executable file. Click Test Connection to make sure your settings ensure successful connection.
8. In the Timeout field, specify the lapse of time in seconds during which IntelliJ IDEA waits for response from the server. If the server does not respond in due time, the user is prompted to disable integration.
9. To enable displaying the branch history of a specified file, including all file branch points, edits, and merges, select the Show branching history check box.
10. To have IntelliJ IDEA point at committed changes that are also integrated to other changelists and provide information on the target changelists that received the content in question, select the Show integrated changelists in committed changes check box.
11. To get the user interface for attaching and detaching Perforce jobs to changelists, select the Enable Perforce Jobs Support check box.

Handling Modified Without Checkout Files

If you are going to modify or delete a file under Perforce version control, the read-only status of such file should be removed. IntelliJ IDEA takes care of automatically making files writable. However, you can change read-only status manually, which may happen in a number of ways; for example:

- With the Clear Read-Only Status dialog enabled , you make a file writable using file system.
- When a read-only file is opened in the editor, you double-click lock icon in the status bar.
- You remove read-only attribute externally, using file properties.

In these cases, the file gets status Modified without checkout and appears in the Local Changes tab of the Version Control tool window .

To resolve 'modified without checkout' files

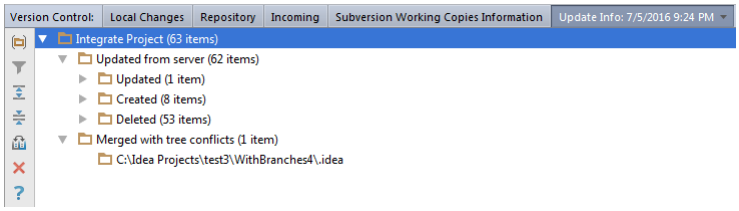
1. In the Local Changes tab of the Version Control tool window , expand Modified without Checkout node, and select the desired file.
2. On the context menu of the file, choose Check Out. The file becomes writable, and moves to the active changelist.

Integrating Perforce Files

You can merge changes between the branches into your local working copy, using the branch specification, or a changelist.

The Integrate Project command is available for both Perforce and Subversion integrations.

Integration results display in the Integrate Info tab of the Version Control tool window. Context menu of a file enables you to compare versions, view history and annotations, browse changes and more.



To integrate a Perforce branch into a project

1. On the main menu, choose VCS | Integrate Project.
2. In the Integrate dialog box, select the Perforce tab (if both Perforce and Subversion integrations are used in this project).
3. Select the sources to be merged, and the desired revision.
4. Define VCS-specific merge options.
5. Click OK.

Resolving Conflicts with Perforce Integration

As described in the section Resolving Conflicts , the conflicts might occur in course of team work. Perforce integration makes use of the following commands:

- Resolve enables you to resolve a conflict to a specific file.
- Resolve All applies to all files in a changelist that have merge status.

To resolve conflicts for the files under Perforce version control

1. In the Local Changes tab of the Version Control tool window , select a conflicting file, or a whole conflict node.
2. On the context menu of the selection, choose Resolve or Resolve All. Files Merged with Conflicts dialog box appears.
3. If you want to accept server version and overwrite your local changes, click Accept Theirs. If you want to force your changes to the repository, click Accept Yours. Clicking Merge opens the merge tool, where you can accept or discard each change individually.
4. Once the conflicts are successfully resolved, commit your local version to the repository.

Showing Revision Graph and Time-Lapse View

Using Perforce integration, you can view the Revision Graph or Time-lapse View for a file without leaving the IDE, provided that `p4v.exe` is installed on your computer.

To show Revision Graph or Time-lapse View for a file

1. Select the desired file in any navigation tool window, or open it in the editor.
2. On the main Version Control menu, or on the context menu of the selection, choose Perforce, and then select Revision Graph, or Time-Lapse View on the submenu.
3. With the first invocation, enter password in the login dialog box.

Using Multiple Perforce Depots with P4CONFIG

If your project contains directories that are stored in the different Perforce depots, you might need to switch between them. IntelliJ IDEA uses P4CONFIG to automatically switch to the respective depot as you use a Perforce-versioned directory.

P4CONFIG is an environment variable that contains the name of P4CONFIG file without a path. If a certain directory is associated with Perforce, IntelliJ IDEA seeks for P4CONFIG file in this directory and its parents; if the file is not found, it is sought in the bin directory of IntelliJ IDEA installation. When a P4CONFIG file is found, IntelliJ IDEA uses the settings contained therein, to connect to the respective Perforce depot.

A sample P4CONFG file might consist of such lines:

```
P4CLIENT=MyClient
P4USER=MySelf
P4PORT=ida:3456
```

To use multiple Perforce depots in a project, follow these general steps

1. Create a P4CONFIG file in each directory associated with Perforce.
2. Create environment variable P4CONFIG that contains file name without a path.

Working Offline

In this section:

- Offline mode basics
- Going offline
- Going online

Offline mode basics

The Perforce plugin keeps a log of VCS operations performed while offline, and replays the log when the user comes back online. The log of operations is stored in the `.iws` file and persists between IntelliJ IDEA restarts.

While offline, you can perform the following operations, which will be automatically replayed in online mode:

- Edit
- Add/Copy
- Delete
- Move/Rename
- Revert
- Move to another changelist
- View Committed/Incoming changes (displaying cached information only).

The following operations are not supported in offline mode: update, commit, integrate, tracking of the unversioned, locally deleted and *modified without checkout* files (unversioned files are shown as *unchanged*), and any other operations that require server connection.

Tip The performance of IntelliJ IDEA Perforce integration in offline mode is considerably better than in online mode (because no server calls are required), so you might want to use offline mode even though connection to the Perforce server is successful.

To go to offline mode, do one of the following

- Automatically, when the Perforce server becomes unavailable. IntelliJ IDEA switches to the offline mode automatically, and displays an offline notification in a pop-up window. To enable this behaviour, select the Switch to offline mode automatically if Perforce is unavailable check box in the Perforce page of the Settings dialog box.

- Manually at anytime, by choosing VCS | Perforce and select Work Offline on the main menu.

When offline mode is activated, the following notification balloon appears:

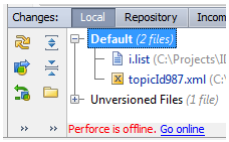
This balloon fades after a while; Perforce is offline message appears at the bottom of the Local Changes tab of the Version Control tool window.

To return to online mode, do one of the following

- Choose VCS | Perforce and clear Work Offline.

- In the offline notification balloon, click the Go online link.

- In the Local Changes tab of the Version Control tool window, click the Go online link:



Checking Perforce Project Status

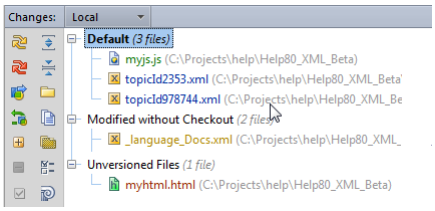
Besides indicating the current file status relative to the repository, IntelliJ IDEA integration with Perforce provides you with the accumulated view of the project files' statuses.

In this section:

- Viewing the statuses of project files
- Refreshing file status

To view differences between the current state of the project files and the repository

1. Open the required project.
2. On the main menu, choose VCS | Refresh File Status.
3. Switch to the Version Control tool window, tab Local .



The status of each file is indicated by the color in which the path to the file is displayed.

To refresh the statuses of project files

IntelliJ IDEA provides two refresh modes for statuses of files under Perforce control.

- Standard Refresh takes into consideration only the changes made through the IntelliJ IDEA integration with Perforce. This improves performance because does not require connecting to the server. However, this approach does not let you know about the changes made outside IntelliJ IDEA, for example, right through the `p4v client` application.
- Force Refresh considers all the changes made to project, both from IntelliJ IDEA and from any other application, for example, right through `p4v client` .

1. Switch to the Version Control tool window, tab Local .
2. Do one of the following:
   - To run Standard Refresh, click the Refresh toolbar button or press `Ctrl+F5` .
   - To run Force Refresh, click the Force Refresh toolbar button .

Attaching and Detaching Perforce Jobs to Changelists

The Perforce integration with IntelliJ IDEA provides you with a user interface for attaching and detaching Perforce jobs to changelists.

Note that the integration does not support creation of Perforce jobs.

To get access to working with Perforce jobs, select the Enable Perforce Jobs Support check box on the Perforce page of the Settings dialog box.

From this topic you will learn how to:

- Attach jobs to changelists:
  - At any stage of your work from the Local tab.
  - During commit from the Commit Changes dialog box.
- Find the desired job to attach to a changelist using:
  - The standard search functionality, which enables you to specify numerous search criteria and thus to flexibly configure the search procedure.
  - The quick search functionality, which enables you to have the found job linked to the changelist automatically.

    Tip This functionality is helpful when you need to attach only one job to a changelist and you either know the exact name of the desired job or at least can specify a search pattern for the name.
- Detach jobs from changelists.

To find and link a job at any stage of your work

1. Open the Local tab of the Version Control tool window.
2. Select the changelist you want to link a job to.
3. From the context menu of the changelist, choose Edit Associated Jobs. The Edit Jobs Linked to Changelist dialog box opens.
4. Find the desired job. Do one of the following:
   - To use the standard search functionality, click the button.
   - To use the quick search functionality, click the + button.
5. View the details of the found job and click OK.

To find and link a job during commit

1. In the Local Changes tab of the Version Control tool window, select the changelist you want to link a job to and open the Commit Changes dialog box.
2. Find the desired job using the controls in the Jobs area. Do one of the following:
   - To use the standard search functionality, click the button.
   - To use the quick search functionality, click the + button.
3. View the details of the found job and continue the commit procedure.

To find a job using the standard search functionality

1. In the Edit Jobs Linked to Changelist dialog box or in the Jobs area of the Commit Changes dialog box, click the button.
   Which dialog box you are currently in depends on whether you are linking jobs during commit or at any other stage of work.
2. In the Link Job to Changelist dialog box that opens, specify the desired search parameters.

   Tip At least one of the fields should be filled in.
3. Click the Search button. The jobs that match the specified criteria are shown in the Search Results list. To view the details of a job, select it in the list.
4. Select the desired job and click OK. The dialog box closes and you return to the dialog box where you started the search:
   - The Commit Changes dialog box, if you are linking jobs during commit .
   - The Edit Jobs Linked to Changelist dialog box, if you are linking jobs at any other stage stage of work.

To quickly find and link one job

1. Open the Edit Jobs Linked to Changelist dialog box or switch to the Jobs area of the Commit Changes dialog box.
   The dialog box to be used depends on whether you are linking jobs during commit or at any other stage of work.
2. In the text box, type the desired job name search pattern and click the + button. The job is found and automatically linked to the current changelist.
   If no job matching the specified pattern is found, the corresponding information message is displayed.

   Warning! The details of jobs that are found and linked through the quick search functionality are available only in the Edit Jobs Linked to Changelist dialog box.

To detach a job from a changelist

- In the Edit Jobs Linked to Changelist dialog box or in the Jobs area of the Commit Changes dialog box, select the desired job and click the − button.
  The dialog box to be used depends on whether you are detaching jobs during commit or at any other stage of work.

Using Subversion Integration

With the Subversion integration enabled, you can perform basic Subversion operations from inside IntelliJ IDEA.

IntelliJ IDEA currently supports integration with Subversion 1.9.

Prerequisites

- IntelliJ IDEA comes bundled with the Subversion plugin. This plugin is turned on by default. If it is not, make sure that the plugin is enabled.
- IntelliJ IDEA Subversion integration does not require a standalone Subversion client. All you need is an account in your Subversion repository.
- Subversion integration is enabled for the current project root or directory.

Subversion Support

- When Subversion integration with IntelliJ IDEA is enabled, the Subversion item appears on the VCS menu, and on the context menus of the Editor and Project views.
- The files in the folders under the Subversion control are highlighted according to their status. See File Status Highlights for file status highlighting conventions.
- Modifications results are shown in the Version Control tool window .
- When using Subversion integration, it is helpful to open the Version Control tool window. The Console tab displays the following data:
  - All commands generated based on the settings you specify through the IntelliJ IDEA user interface.
  - Information messages concerning the results of executing generated Subversion commands.
  - Error messages.

Authenticating to Subversion

The Subversion server does not require user authentication on every request. When you use Subversion integration in IntelliJ IDEA, you only need to answer the authentication challenge of the server if it is required by the authentication and authorization policies. Upon successful authentication, your credentials are saved on disk, in `~/.subversion/auth/` on Unix systems or `<USER HOME>/.subversion_IDEA` on Windows and OS X.

When an authentication challenge comes from the server, the credentials are sought for in the disk cache; if the appropriate credentials are not found, or fail to authenticate, you are prompted to specify your login and password.

If necessary, you can opt to delete all credentials stored in the cache for the http, svn and ssh+svn protocols.

To delete credentials from disk

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Version Control.
2. Open the Subversion settings page and click the Clear Auth Cache button.

Browsing Subversion Repository

Prior to checking files out, you can browse the contents of a Subversion repository. The Subversion Repository browser enables you to add or discard repository locations, view the history of files and folders, check out files and folders, navigate to the source code, browse changes, create branches or tags, etc.

Browsing the contents of a Subversion repository is always available, even when Subversion is not enabled in your project. All you need is a valid user account.

To browse the contents of a Subversion repository

1. On the main menu, choose VCS | Browse VCS Repository | Browse Subversion Repository.
2. The SVN Repositories tool window will open.
3. If no repositories have been specified so far, click + in the toolbar and specify the repository URL in the New Repository Location dialog that opens.
4. Browse the repository. If authentication is required, enter your credentials in the Authentication Required dialog that opens when you first try to expand the top node.

Checking Out Files from Subversion Repository

By checking out files from a Subversion repository, you obtain a local working copy of the repository , which you can edit. After making the necessary changes, you can publish the results by committing, or checking in your changes to the repository.

To check out files from a Subversion repository, do the following:

1. On the main menu, choose VCS | Checkout from Version Control | Subversion.
2. In the Check Out From Subversion dialog box, expand the desired repository location and select the element you want to check out.
3. Click the Checkout button.
4. In the dialog that opens , specify the destination directory where the local copy of the repository files will be created, and click OK.
   If you are checking out sources for an existing project, the destination folder should be below the project content root .
5. In the SVN Checkout Options dialog box, specify the following settings:
   - Revision to be checked out (HEAD or a selected revision).
   - Whether you need to check out the nested directories.
   - Whether you need to include the external locations.

   Click OK.

Tip This action is also available from the SVN Repositories tool window . Right-click a directory and choose the required command from the context menu.

IntelliJ IDEA suggests to create a project based on the sources checked out from version control.

If you accept the suggestion, the New Project from Existing Code Wizard starts.

Cleaning Up Local Working Copy

The Cleanup command in Subversion can be helpful in the following situations:

- Your local working copy is in an inconsistent state because a Subversion command was interrupted.
- The timestamp of a file has changed while its content remains intact.

To clean up the local working copy, do one of the following:

- Select the desired file or directory in the Project tool window and choose Subversion | Cleanup from the context menu of the selection.
- Open the desired file in the editor and choose VCS | Subversion | Cleanup from the main menu.
- Select the desired file or directory in the Local Changes tab of the Version Control tool window and choose Subversion | Cleanup from the context menu of the selection.

Comparing With Branch

In addition to the common file versions comparison options, the Subversion integration with IntelliJ IDEA provides a special command that enables you to compare a file from your local working copy with its version in the selected branch.

To compare a file with its version in a specified branch, do the following:

1. Select the desired file in the Project tool window , or open it in the editor.
2. From the main VCS menu, or on the context menu of the selection, choose Subversion | Compare with Branch.
3. In the Compare with Branch pop-up, select the desired branch. The Compare with Branch dialog in the form of the Differences viewer for files appears.

Tip This action is also available in the SVN Repositories browser. Right-click the desired directory and choose the corresponding command from the context menu.

Configuring the Format of the Local Working Copy

A working copy is a directory that contains a collection of files which you can use as your private work area, as well as some extra files, created and maintained by Subversion. For example, each directory in your working copy contains an administrative directory named `.svn`.

You can have local working copies created with Subversion 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, or 1.9. IntelliJ IDEA handles all these formats, giving you a choice to upgrade to the new format or preserve the legacy one.

Switching to another Subversion format is always associated with a specific working copy (directory) under the Subversion control. In other words, one cannot upgrade to a newer Subversion format at the IDE level, but only to a directory under Subversion control.

Tip To check VCS association, go to File | Settings | Version Control where all mappings between the project directories (or the entire project) and VCSs are listed.

To change the format of the local working copy, do the following:

1. Open the Version Control tool window by doing one of the following:
   - On the main menu, choose View | Tool Windows | Version Control.
   - Press `Alt+9`.
2. Switch to the Subversion Working Copies Information tab.
   This tab is only available, when the current project sources are entirely or partially under Subversion control.
3. Scroll to the information on the required directory and click the Change link.

   Note Note that Subversion 1.9 can be used with the local working copy version 1.8, so in this case the Change link will not appear.
4. In the Convert Working Copy Format dialog box, that opens, select the desired format option.

Configuring HTTP Proxy

Subversion stores the http proxy settings in the `servers` file in the user's runtime configuration area (`~/.subversion/auth/` on Unix systems or `<USER HOME>/Application Data/Subversion/auth/` on Windows).

You can configure the Subversion proxy settings in two ways:

- Edit the servers file manually.
- Configure proxy settings directly from IntelliJ IDEA. Do the following:
  1. In the Settings dialog, go to Version Control | Subversion and open the Network tab.
  2. Click the Edit Network Options button and specify the proxy settings in the Edit Subversion Options Related to Network Layers dialog box that opens.

Configuring Subversion Repository Location

Subversion repository locations are global IntelliJ IDEA properties. It means that the configured repository locations will be available no matter if a project is open or not, which is useful if you need to check out an entire project from Subversion. You can define multiple Subversion repository locations for future use.

To configure a Subversion repository location, do the following:

1. Open the SVN Repositories tool window by choosing VCS | Browse VCS Repository | Browse Subversion Repository from the main menu.
2. In the SVN Repositories tool window choose New | Repository Location from the context menu, or click the + button on the toolbar.
3. In the New Repository Location dialog, specify the repository URL.

Configuring Subversion Branches

IntelliJ IDEA allows you to compose a list of parent folders of the branches you work with. This list will be displayed every time you perform any operation with branches, for example, when you synchronize your local working copy, compare branches, etc.

Branches are configured in the Configure Subversion Branches dialog box.

To configure Subversion branches, do the following:

1. Acc the Configure Subversion Branches dialog : in the Version Control tool window, switch to the Subversion Working Copies Information tab, and then click the Configure Branches link.
2. In the Trunk location field, specify the URL of your repository trunk. Type the address or click the Browse button ... and select the trunk location in the Select Repository Location dialog that opens. This dialog shows a tree of all branches and tags under the repository root .
3. In the Branch locations area, make up a list of folders where the branches you need in your work are stored. Use the + and – buttons to add/remove branches to/from the list.

Creating Branches and Tags

IntelliJ IDEA allows you to create branches or tags on the basis of your local working copies, or their repository versions.

To create a branch or a tag in a Subversion repository, do the following:

1. From the main menu, choose VCS | Subversion | Branch or Tag. Alternatively, select the source folder in the SVN Repositories tool window and choose the Branch or Tag command from the context menu.
2. In the Create Branch or Tag dialog that opens, in the Copy From section, specify the source folder that will be copied to a branch or a tag. You can use your local working copy, or a repository location.
   If a repository location is selected as the source:
   - Click to fill the Repository Location field with the path to the project location.
   - Specify the revision to base the new branch on. This can be the HEAD revision, or a revision with the specified number. If the Specified option is selected, type the revision number, or click and find the desired revision in the Subversion Changes Browser.
3. In the Copy To section, specify the destination where the branch or the tag will be created. If you use the base URL, specify the name of the new branch or tag. If you opt to create a branch or tag in another repository location, type its URL, or click and select the destination from the Select Repository Location dialog.
4. Optionally, enter a comment a click OK.

Exporting Information From Subversion Repository

You might need to obtain a *clean* local copy of the Subversion working tree without the `.svn` catalogs. Instead of checking files out and then manually deleting the administrative directories, you can use the Export command available in the Subversion repository browser.

To export a directory from a Subversion repository, do the following:

1. In the main menu, select VCS | Browse VCS Repository | Browse Subversion Repository to open the the SVN Repositories tool window.
2. Right-click a directory you want to export and choose Export from the context menu.
3. In the Select Path dialog that opens, specify the destination directory and click OK.
4. In the SVN Export Options dialog that opens, check the Export and Destination paths and specfiy the following options:
   - Depth: use this drop-down list to specify the range of recursion into Subversion subdirectories. The available options are:
     - working copy: select this option to get files/directories from the repository subtrees that have not been checked out yet.
     - empty: select this option to involve only the current file.
     - files: select this option to involve files from the current folder.
     - immediates: select this option to involve direct children of the current file.
     - infinity: select this option to enable full recursion.
   - Replace existing files: select this option to replace files in the destination directory with the exported sources.
   - Include external locations: select this option to include external references into the export.
   - Override 'native' EOLs with:: use this drop-down list if you want to override the `svn:eol-style=native` property. This is useful if team members sharing the same repository use different operating systems, which may result in problems with line endings. The following options for line separators are available:
     - None: this option is selected by default, and keeps the `svn:eol-style=native` property unchanged.
     - LF: select this option if you are using unix
     - CRLF: select this option if you are using Windows
     - CR: select this option if you are using OS X

Importing a Local Directory to Subversion Repository

You can import an entire directory to your Subversion repository provided that you have access rights. This is helpful for putting a whole project under version control.

Import to the repository is always available, even if Subversion integration is not enabled for your project.

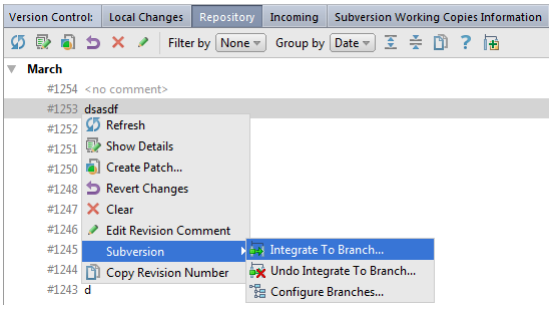
To import a directory to a Subversion repository, do the following:

1. From the main menu, choose VCS | Import into Version Control | Import into Subversion.
2. In the Import Into Subversion dialog that opens, select the target Subversion repository. If the desired target repository location is not in the list, click the + button to add it (for more details, see Configuring Subversion Repository Location). Click Import.
3. In the Select Path dialog that opens, specify the directory you want to import and click OK.
4. In the SVN Import Options dialog that opens, check the Import to and Import from paths and specfiy the following options:
   - Depth: use this drop-down list to specify the range of recursion into Subversion subdirectories. The available options are:
     - working copy: select this option to get files/directories from the repository subtrees that have not been checked out yet.
     - empty: select this option to involve only the current file.
     - files: select this option to involve files from the current folder.
     - immediates: select this option to involve direct children of the current file.
     - infinity: select this option to enable full recursion.
   - Include ignored resources: select this option to include files that have been added to the Subversion ignore list and are not subject to version control into the import.
5. Optionally, enter a commit message, or select one from the Recent Messages drop-down list and click OK.

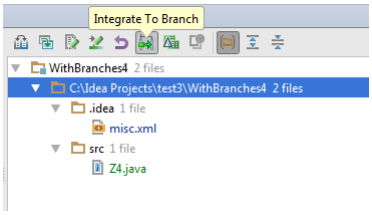
Integrating Changes to Branch

IntelliJ IDEA allows you to integrate changes into a selected branch, and commit the results of such integration to the repository. Do the following:

1. In the Version Control tool window, switch to the Repository tab .
2. Right-click the changelist you want to integrate, and select Subversion | Integrate to Branch from the context menu:



If you want to integrate separate files instead of the whole changelist, select them in the Changed Files pane and choose Subversion | Integrate To Branch from the context menu:



The Integrate to Branch dialog opens displaying the URL addresses of the source and target branches and a list of available local working copies.

3. From the Integrate into working copy list, select the path to the local working copy into which the selected changelist will be integrated.
   To add a path to the list, click the + button.

   To remove a path from the list, click the — button.

   Note Make sure the specified working copy directory is under Subversion version control!

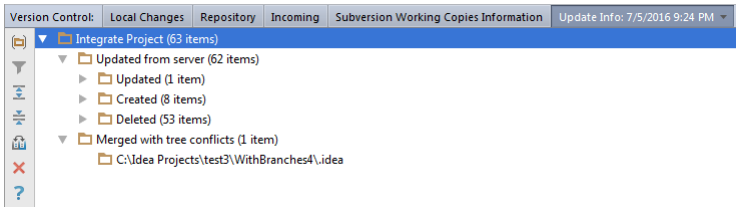
4. To preview the merge result by enabling the `--dry-run` switch of the `svn` command, select the Try merge, but make no changes check box.
   If this check box is not selected, sources are merged silently.
5. Click OK. The Commit Changes dialog opens.
6. Review the summary, specify the necessary options, and run commit .

Integrating SVN Projects or Directories

Integrating projects or directories in Subversion means merging the differences between the two specified revisions into your working copy.

The Integrate Project command is available for both Subversion and Perforce.

Integration results are displayed in the Update Info tab of the Version Control tool window . The context menu of a file allows you to compare versions, view file history and annotations, browse changes, etc.



To integrate different sources into one Subversion project, do the following:

1. From the main menu, choose VCS | Integrate Project. The Integrate Project dialog opens.
2. If both Subversion and Perforce are used as version control systems in your project, select the Subversion tab.
3. In the Source 1 and Source 2 fields, specify the sources to be merged and select the revision. If you check the Specified option, you can click the Browse button and select a revision from the Changes Browser.
4. If necessary, select the following merge options and click OK:
   - Use ancestry: if this option is selected, ancestry of files will be noticed (this corresponds to the `svn merge` command). If unchecked, any relations between files and directories will be ignored (corresponds to `svn diff`).
   - Try merge but make no changes: select this option to preview merge results by enabling the `--dry--run` option of the SVN command. If unchecked, sources will be merged silently.
   - Depth: use this drop-down list to specify the range of recursion into Subversion subdirectories. The available options are:
     - working copy: select this option to get files/directories from the repository subtrees that have not been checked out yet.
     - empty: select this option to involve only the current file.
     - files: select this option to involve files from the current folder.
     - immediates: select this option to involve direct children of the current file.
     - infinity: select this option to enable full recursion.

Integrating Changes To/From Feature Branches

A Feature Branch is intended for working on one particular feature. It is normally constituted of data downloaded from the trunk and is integrated back into the trunk when the work on the feature is completed. You can apply all the changes or select a subset of them. IntelliJ IDEA creates a changelist with the merged changes and offers it for commit.

To integrate changes from a branch

1. Open the Version Control tool window and switch to the Subversion Working Copies Information tab .
2. Click the Merge from link, and choose the changes' source on the context menu. The available options are:
   - trunk - choose this option to merge the changes from the trunk to the current branch.
   - branches - choose this option to have changes from a specific branch applied to the current branch. Select the source branch in the Branches pop-up list.
   - tags - choose this option to have changes from a specific branch applied to the current branch. Select the source branch in the Tags pop-up list.

   Tip To compose a list of branches to work with, choose the Configure branches option and update the list of branches in the Configure Subversion Branches dialog box that opens.
3. In the Merge from <branch_name> dialog box that opens, specify the scope of changes to apply.
   - To have all changes applied, click the Merge all button.
   - To have a subset of changes applied, click the Select revisions to merge button. In the list of revisions that is displayed, appoint the revisions to apply changes from by selecting check boxes next to the desired revisions. To have the selected changes applied, click the Merge selected button.

     Tip To have all changes applied regardless of the selection, click the Merge all button.
4. In the Files merged with conflicts, view the list of files where problems occurred during the merge procedure and resolve the problems using the following buttons:
   - Accept Yours - click this button to have IntelliJ IDEA force your changes.
   - Accept Theirs - click this button to have IntelliJ IDEA overwrite your changes.
   - Merge - click this button to open the merge tool and resolve the conflicts there.

Locking and Unlocking Files and Folders

Though Subversion integration allows you to successfully modify and merge files changed by different team members, sometimes it makes sense to lock files (for example, images) to avoid overwriting changes.

To lock a file:

1. Select the file you want to lock in the Project tool window or open it in the editor.
2. From the main menu, select VCS | Subversion | Lock from the main menu, or Subversion | Lock from the context menu of the selection.
3. Specify the lock name in the Lock File dialog that opens.

To unlock a file, select the file you want to unlock, or open it in the editor, and choose VCS | Subversion | Unlock from the main menu, or Subversion | Unlock from the context menu of the selection.

Tip To forcibly break the lock set by somebody else, select the Steal Existing Lock check box.

Resolving Text Conflicts

If a conflict occurs in a file under the Subversion version control, conflict markers are added to the conflicting file, and three auxiliary unversioned files are created in your local working copy:

- filename.mine: the copy of your local file without conflict markers.
- filename.rOld: the base revision you have last synchronized to.
- filename.rNew: the latest version on the server.

Conflicting files are marked with red in the Local Changes tab of the Version Control tool window. In the Update Info tab, they are grouped in the Merged with conflicts list and are also marked with red.

With IntelliJ IDEA, you can resolve conflicts in two ways:

- Semi-automatically, using a merge tool.
- Manually in the editor. After that, you need to manually mark the processed files as conflicts-free.

In this section:

- To resolve a text conflict using the merge tool
- To resolve a text conflict manually
- To mark a file as resolved

To resolve a text conflict using the merge tool

1. In the Local Changes tab of the Version Control tool window, select the conflicting file:

2. On the main VCS menu, or on the context menu of the selection, choose Subversion | Resolve Text Conflict. The Files Merged with Conflicts dialog appears.
3. If you want to accept the server version and overwrite your local changes, click Accept Theirs. If you want to force your changes to the repository, click Accept Yours. Clicking Merge opens the merge tool, where you can accept or discard each change individually. As a result, the file is automatically marked as resolved, and auxiliary files are deleted.
4. Once the conflicts have been successfully resolved, commit your local version to the repository.

To resolve a text conflict manually

Open the conflicting file in the editor, and do one of the following:

- Edit the contents within the conflict markers as required.
- Copy one of the auxiliary files on top of your working file.

To mark a file as resolved

Do one of the following:

1. Do one of the following:
   - Select the file in the Project tree or in the Local Changes tab of the Version Control tool window, and choose Subversion, and then choose Mark Resolved on the context menu of the selection.
   - With the conflicting file opened in the editor, right-click the mouse anywhere in the editor tab. On the context menu choose Subversion, and then choose Mark Resolved.
   - On the main menu, choose VCS | Subversion | Mark Resolved..
2. In the Mark Resolved dialog box that opens select the file in question.
3. Click the Mark Resolved button.

Sharing Directory

IntelliJ IDEA supports the useful sharing directories feature of Subversion. The Share Directory command applies to the content roots. When such content root is shared, a new directory is created in the specified location of the Subversion repository, checked out to the local directory you want to share, and the contents of the local directory is added to the repository. After that the local directory contains a valid working copy of the repository.

To share a directory via Subversion repository

1. In the Project tool window, select an unversioned directory, which is a content root of a module associated with Subversion.
2. On the main menu, choose Subversion | Share Directory.
3. Specify the target repository location, where the directory should be shared.

Viewing and Fast Processing of Changelists

If you use Subversion 1.5 or higher on the server and in your local working copy, you can derive advantage from the extended Merge Info functionality implemented through the Merge Info pane of the Version Control tool window, Repository tab.

With this functionality, instead of browsing all changelists within a certain period, you can define a set of changelists to display. This is done by specifying a pair of branches in the repository (source and target) whereupon IntelliJ IDEA shows only those changelists from the source branch that have been integrated into the target branch.

Additionally, you can specify various filtering options to minimize the number of extraneous changelists.

Finally, integration and managing integration status are also available directly from the Merge Info pane.

The extended browsing functionality includes:

- Defining the Set of Changelists to Display
- Filtering Out Extraneous Changelists
- Integrating Files and Changelists from the Version Control Tool Window
- Viewing and Managing Integration Status

Tip Before enabling extended Merge Info functionality, make sure you are using SVN Server 1.5 or higher.

To enable Merge Info functionality

1. To use SVN 1.5 format in the local working copies, choose VCS | Subversion in the main menu, then choose Show Working Copies in the submenu.
2. In the Subversion Working Copies Information dialog box that opens, select the relevant working copy and click the Change Working Copy Format button.
3. In the Convert Working Copy Format dialog box, choose the 1.5 format option and click OK.
4. To have the Merge Info pane displayed, switch to the Version Control tool window, Repository tab, and click the Highlight Integrated button on the toolbar.
5. On the Merge Info pane, configure the source and target branches.

Defining the Set of Changelists to Display

With the extended Merge Info functionality, you can restrict the set of changelists displayed in the Changelists pane to the changelists that have been integrated from one specific branch to another. These branches are referred to as "source" and "target" respectively.

To specify the branches involved

1. Open the Version Control tool window and switch to the Repository tab.
2. To have the Merge Info pane displayed, click the Highlight Integrated button on the toolbar.
3. In the From text box, specify the URL address of the source branch.
4. In the To text box, specify the path to the target branch.
   If necessary, use the Browse button to open the Select Branch dialog box.
5. Specify the path to the local working copy to which you will apply patches created based on selected changelists.
   If necessary, use the Browse button to open the Configure Working Copy Paths dialog box.

Filtering Out Extraneous Changelists

With the extended Merge Info functionality, you can make the set of changelists displayed in the Changelists pane even easier to browse by applying additional filtering according to integration status, committer, or version control system used.

Tip Filtering out integrated/not integrated changelists is available only when integration status is highlighted.

Otherwise the Filter out integrated button and Filter out not integrated button are unavailable. To enable integration status highlighting, click the Highlight Integrated button on the toolbar.

To apply additional filtering options, do one of the following in the Merge Info pane

- To display only changelists that have not been integrated into the working copy, click the Filter out integrated button on the toolbar.
- To display only changelists that have been integrated into the working copy, click the Filter out not integrated button on the toolbar.
- To hide changelists that are managed in another VCS or are located under another root, click the Filter out others button on the toolbar.
- To display only the changelists that were committed by a specific user, select User in the Filter by drop-down list. Then select the name of the required user.
- To display only the changelists applied to a specific module or folder, select Structure in the Filter by drop-down list. Then select the required location.
- To have the changelists grouped by users who committed them or by commit dates, select the relevant option in the Group by drop-down list.

Integrating Files and Changelists from the Version Control Tool Window

You can integrate changelists or files into your local working copy directly from the Version Control tool window.

To integrate a changelist or a file into your working copy, complete the following general steps

- In the Changelists pane, select the required changelist.
  If necessary, you can select and integrate several changelists at a time.
- To integrate the entire changelist, click the Integrate to Branch button on the Merge Info pane toolbar.
- To integrate a particular file from the selected changelist, select the file in the Changed Files pane and click the Integrate to Branch button on the Merge Info pane toolbar.
- To revert the last integration of the selected changelist into the working copy, click the Undo Integrate to Branch button on the toolbar.

Viewing and Managing Integration Status

With the extended Merge Info functionality, you can view and update integration status of changelists.

Subversion stores information on whether a changelist has been integrated into the local working copy or not. Based on this data, IntelliJ IDEA informs you on the integration status of a specific changelist by displaying one of the following icons next to the changelist:

- integrated
- not integrated
- integration status unknown
- common history

To have the integration status displayed, click the Highlight Integrated button on the Merge Info pane toolbar.

You can change the integration status of a changelist without actually integrating it into the working copy or reverting the previous integration. This will affect only the administrative data.

To toggle the integration status of a changelist

Do one of the following:

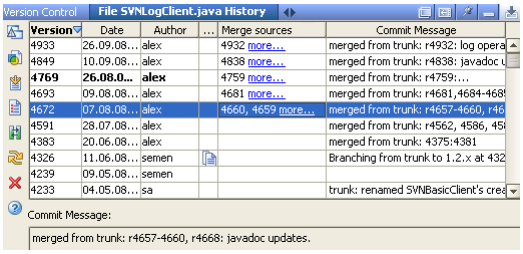
- To mark a changelist as Integrated, select the changelist and click the Mark As Merged button on the Merge Info pane toolbar.
- To mark a changelist as Not integrated, select the changelist and click the Mark As Not Merged button on the Merge Info pane toolbar.

Viewing Merge Sources

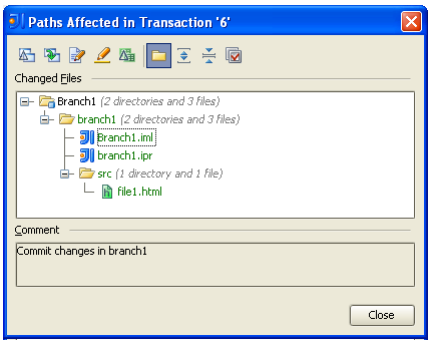
For Subversion, the common Viewing Change History functionality is expanded with the possibility to view which revisions have been used in merges applied to a file. The merge sources can be presented as a tree of revisions.

To view the merge source

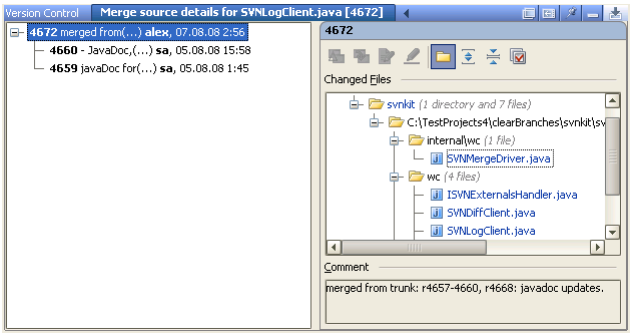
1. Open the Version Control tool window, Repository tab, and switch to the Changelists pane.
2. Select the required changelist, switch to the Changed Files pane, and select the required file.
3. Select Show History in the context menu or click on the main editor toolbar. The Version Control tool window opens displaying the change history of the selected file. For each revision, the Merge sources read-only field shows the revisions that were used as sources for the merge.



4. To view all the paths affected by a merge, select the relevant row and select Show all paths affected in the selected transaction in the context menu or click on the tool window toolbar.
   The following information dialog box appears:



5. To view the source details of a merge, select the relevant row and click More. The details are displayed in the following information dialog box:



Working With Subversion Properties for Files and Directories

Subversion integration enables you to work with Subversion-specific properties without leaving IntelliJ IDEA.

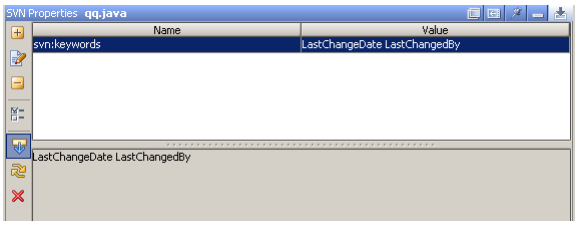
Once defined, properties of a file or directory display in the SVN Properties view. In this view you can explore and change existing properties and their values, or create new ones, using the toolbar buttons or context menu commands.

This section describes how to:

- View properties of a file or directory from within IntelliJ IDEA.
- Create a new property, or change the value of an existing property .
- Set up a keyword property.
- Delete a property .
- Resolve property conflicts .

To view the properties of a file or directory

1. In the Project tool window , select the desired file or directory under SVN version control.
2. On the main VCS menu, or on the context menu of the selection, choose Subversion | Edit Properties. SVN Properties view displays properties of the selected file.



3. Use the toolbar buttons or context menu commands to create, edit or delete properties, as described in the procedures below.

Tip If you want SVN Properties view to preserve its contents as you navigate through your project or edit files, make sure that the Follow Selection button is not pressed; otherwise the view will show properties for the currently selected or edited file.

To create a new property, or set up the value for an existing property

1. Open SVN Properties view .

   Tip Use Set property command on the Subversion menu to define a single property.

2. Click add button + on the toolbar of the SVN Properties view, or choose the Add property command on the context menu. The Set Property dialog box appears.
3. In the Property name field, type the name of the new property, or select one from the drop-down list.
4. Choose the Set property value option, and specify the desired value in the text area below.
5. To have the changes applied to all subdirectories of the selected directory, select the Update properties recursively check box.
6. Click OK.

To set up svn:keywords property

1. In the SVN Properties view for a file, click .
2. In the SVN Keywords dialog box, check the keywords to be included in the property.
3. Click OK.

To delete a property in the SVN Properties view

1. Select a property to be deleted.
2. Click − on the toolbar.

To delete a property in the Set Property dialog box

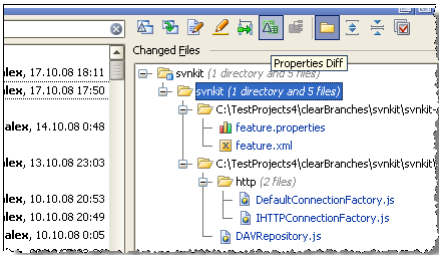
1. In the Property name field, select the property to be deleted.
2. Click Delete property radio-button.
3. If you want this property to be deleted from all files and subdirectories of the selected directory, check the option Update properties recursively.
4. Click OK.

Viewing Differences in Properties

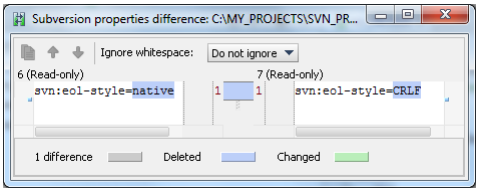
With IntelliJ IDEA, you can view differences in properties between a file in the local copy and in the repository or between two revisions of a file in the local copy.

To view property difference between the local copy and the repository version

1. Open the Version Control tool window and switch to the Repository tab.
2. In the Changed Files pane, select the file for which you want to view property differences.
3. Choose Properties Diff on the context menu of the selection or click on the toolbar.

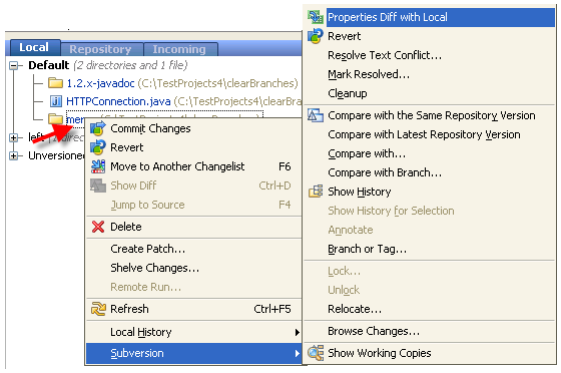


4. In the Subversion properties difference viewer, explore the differences:

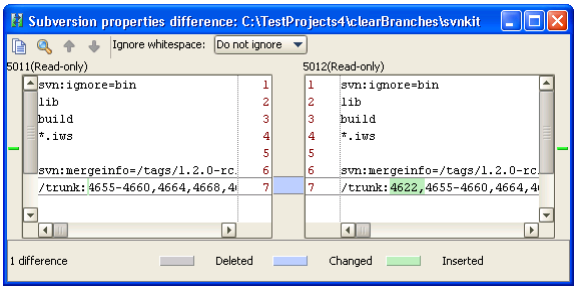


To view property difference between two revisions in the local copy

1. Open the Version Control tool window and switch to the Local tab.
2. Select the file for which you want to view property differences between revisions and choose Subversion | Properties Diff with Local in the context menu.



3. In the Subversion properties difference viewer explore the differences:



Resolving Property Conflicts

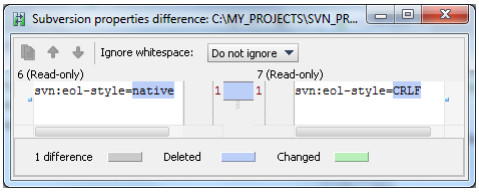
A property conflict is reported during synchronization with the server when IntelliJ IDEA detects differences between the properties of a local file or folder and their version on the server. IntelliJ IDEA does not attempt to resolve property conflicts automatically and displays the files and folders with such conflicts under the Merged with property conflicts node in the Update Info tab of the Version Control tool window. You have to resolve property conflicts manually and then tell IntelliJ IDEA to treat the corresponding files and folders as conflict-free.

On this page:

– Resolving a property conflict
– Marking a file as resolved

To resolve a property conflict

1. Open the Version Control tool window and switch to the Repository tab.
2. In the Changed Files pane, select the conflicting file.
3. Choose Properties Diff on the context menu of the selection or click on the toolbar.
4. In the Subversion properties difference viewer, explore the differences:



5. Based on the detected discrepancies, update the properties so the conflict is resolved.

To mark a file as resolved

1. In the Update Info tab of the Version Control tool window, select the fixed file under the Merged with property conflicts node. As you can see, the file is still displayed in red as conflicting.
2. On the context menu of the selection, choose Subversion, and then choose Mark Resolved. When the dialog box is closed, the Local Changes tab of the Version Control tool window shows the affected files as updated. and available for checking in to the server.
3. Check in the resolved files.

Diagnosing Problems with Subversion Integration

On this page:

- Basics
- VCS and its general settings
- Subversion settings
- The format of the local working copy
- Parent folders of the branches used
- Enabling svnkit logging

Basics

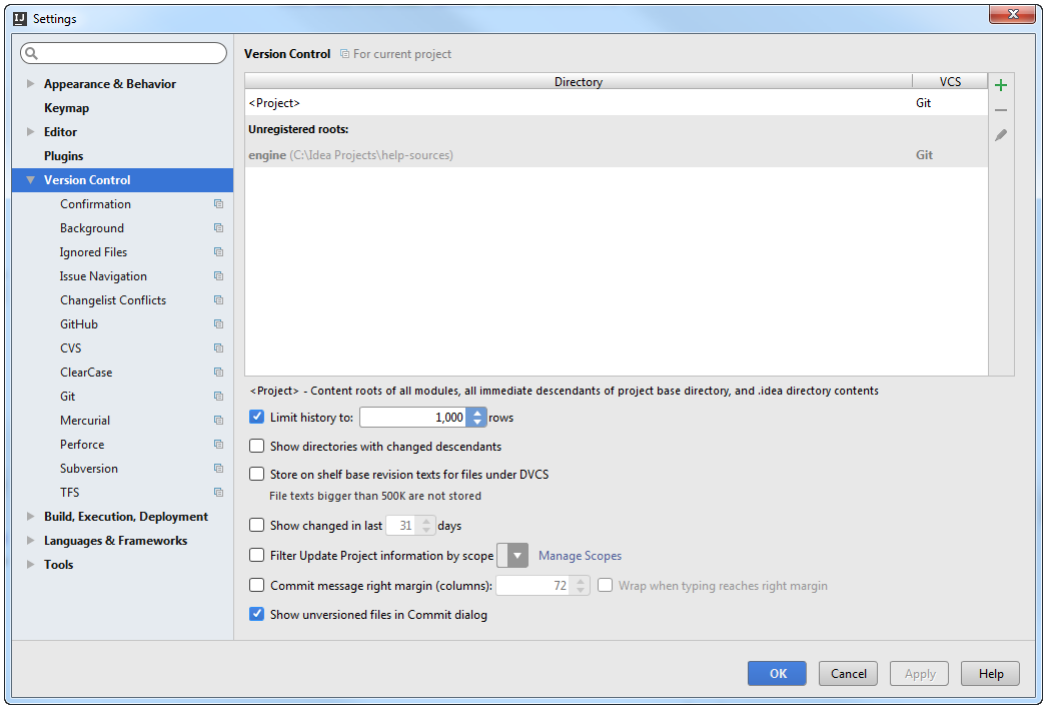
If any problem occurs with Subversion integration, feel free to contact our support. To facilitate detecting, locating, and resolving your problems, provide detailed information regarding your Subversion integration with IntelliJ IDEA. This topic lists the necessary information and explains how you can retrieve it.

The following data is usually required to diagnose Subversion problems:

- The IntelliJ IDEA version and the build number.
- The operating system used.

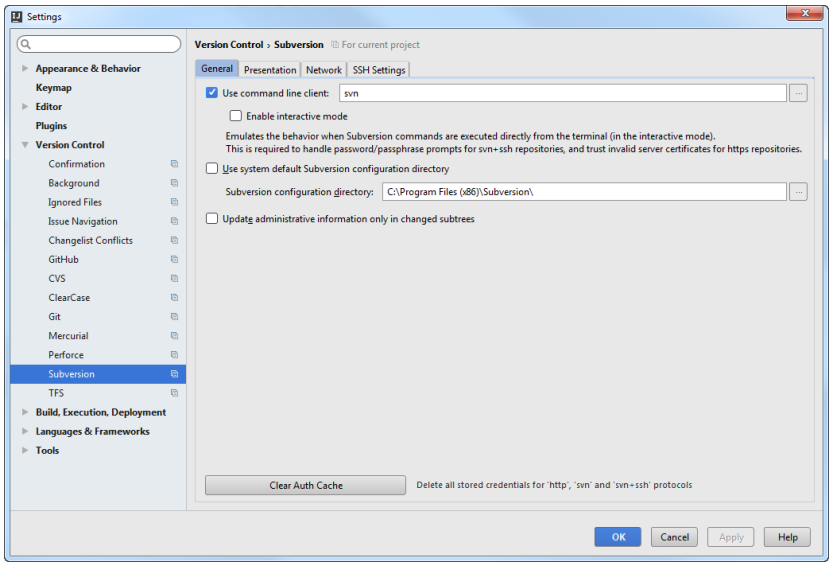
VCS and its general settings

The version control system used in your project or in its parts and the general version control settings applied to your project are specified on the Version Control page of the Settings/Preferences dialog. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Version Control.



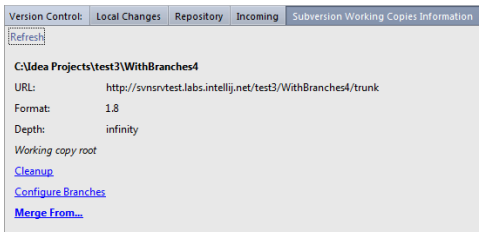
Subversion settings

Subversion-specific settings are configured on the Subversion page, under the Version Control node of the Settings/Preferences dialog. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Subversion under Version Control.



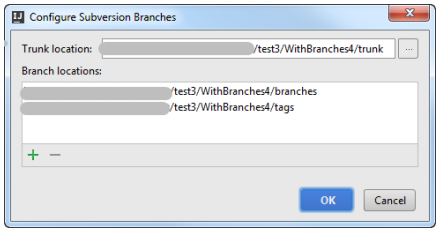
The format of the local working copy

The working copy format is the Subversion format in compliance with which the working copy was created. To view the format of a working copy, choose View | Tool Windows | Version Control on the main menu. In the Version Control tool window that opens, switch to the Subversion Working Copies Information tab.



Parent folders of the branches used

To view the configuration of branches, in the Subversion Working Copies Information tab of the Version Control tool window, click the Configure Branches link.



Enabling svnkit logging

If data saved in the IntelliJ IDEA logs is not sufficient to solve the problem and the problem is related to the communication protocol or authentication, enable svnkit logging, reproduce the problem, and attach the `idea.log` .

To enable svnkit logging, add `-Djavasvn.log=true` to one of the following files depending on the operating system used:

- Windows: `IntelliJ IDEA.exe.vmoptions`
- Linux: `IntelliJ IDEA.vmoptions`
- OS X: http://stackoverflow.com/a/13581526/72788

Refer to sections Tuning IntelliJ IDEA to learn about IntelliJ IDEA `idea.properties` and `vmoptions` files locations.

Using TFS Integration

With the TFS integration enabled, you can perform basic TFS operations from inside IntelliJ IDEA.

IntelliJ IDEA supports TFS up to TFS 2015.

The information provided in this section assumes that you are familiar with the basics of the TFS version control system.

Before You Start

- The TFS integration does not require a standalone TFS client. All you need is an account in your Team Foundation Server.
- Make sure that the TFS Integration plugin is enabled.
  This plugin is bundled with IntelliJ IDEA and enabled by default. If not, enable the plugin .
- Make sure that TFS integration is enabled for the entire project root or a specific directory. For more details, see Enabling Version Control .

TFS Support

- When TFS integration with IntelliJ IDEA is enabled, the TFS item appears on the VCS menu, and on the context menus of the Editor and Project views.
- The files in the folders under the TFS control are highlighted according to their status. See File Status Highlights for file status highlighting conventions.
- Modifications results are shown in the Version Control tool window .
- When using TFS integration, it is helpful to open the Version Control tool window. The Console tab displays the following data:
  - All commands generated based on the settings you specify through the IntelliJ IDEA user interface.
  - Information messages concerning the results of executing generated TFS commands.
  - Error messages.
- TFS integration with IntelliJ IDEA allows you to resolve/associate work items with the new changeset when committing your changes. To achieve this, you can access saved queries and browse work item trees.

In this section:

- Creating and Managing TFS Workspaces
- Checking Out from TFS Repository
- TFS Check-in Policies

Creating and Managing TFS Workspaces

The interaction between your TFS server and local projects is configured through workspaces . A workspace mainly maps the folders in the repository with their copies on your machine.

In IntelliJ IDEA, you can configure access to several TFS servers and have as many workspaces under them as you need. The list of available TFS server access configurations and workspaces is handled through the Manage TFS Servers and Workspaces dialog box.

In this topic:

– Opening the Manage TFS Servers and Workspaces dialog box
– Configuring access to a TFS server
– Creating a workspace

To open the Manage TFS Servers and Workspaces dialog box

1. Open the Settings dialog box , and then click TFS under the Version Control node.
2. On the TFS page that opens, click the Manage button in the Servers and Workspaces area. The Manage TFS Servers and Workspaces dialog box shows the list of all available servers and workspaces in them.

To configure access to a TFS server

1. Open the Manage TFS Servers and Workspaces dialog box with the list of all available servers and workspaces in them.
2. Click the Add button in the Team Servers area.
3. In the Add Team Foundation Server dialog box, that opens, specify the URL address of the target server in the Address text box.
   In the Auth field, specify the authentication protocol to access the server. TFS uses NTLM authentication , so native Windows applications (that is, Microsoft Team Explorer) authenticate silently with system credentials. IntelliJ IDEA users must always specify their username and password because of limitations posed by Java Runtime.

   To authenticate through OAuth (Windows Live ID), choose Alternate from the Auth drop-down list.
4. For the NTLM and Alternate authentication types, specify your credentials:
   – For NTLM: the network domain where the TFS server is located, your TFS user name, and your TFS password.
   – For Alternate: your TFS user name and password.
5. Click OK. IntelliJ IDEA returns to the Manage TFS Servers and Workspaces dialog box, where the new server is added to the list.

To discard a server access configuration, select the server in the list and click the Remove button in the Team Servers area.

To create a server workspace

A workspace is identified by its name and the name of its owner, contains the name of your machine, the URL address of the server the workspace belongs to, and a set of mappings between remote and local working folders that are accessible through the workspace.

1. Open the Manage TFS Servers and Workspaces dialog box, and select the server in question. To refresh the list of the available server workspaces, click the Reload workspaces button.
2. Click the Create button in the Workspaces area.
3. In the Create Workspace dialog box, that opens, specify the workspace name. Optionally, provide a brief description of the workspace in the Comment text box.
4. In the Working folders area, define the mappings.
   1. Click the Add button . A new line is added to the list of mappings.
   2. In the Server path text box, specify the folder on the server you need to work with.
   3. In the Local path text box, specify the local folder to store the downloaded data in.
   4. Specify the status of the mapping in the Status drop-down list.
      – To enable retrieving data from the server according to the mapping, choose Active.
      – To prevent downloading data from the server according to the mapping, choose Cloaked.
   5. To discard a mapping, select it in the list and click the Remove button .

Checking Out from TFS Repository

Checkout helps you obtain a local working copy of the repository folders, which you can edit as required. After making the necessary changes, you can publish results by committing, or checking in changes to the repository.

Interaction between your TFS server and the local projects is configured through workspaces that map the folders in the repository with their copies on your machine. You can have remote folders downloaded either according to the mappings from an existing workspace or define the required mappings during download and have the corresponding workspace generated automatically. In either case, the Checkout from TFS wizard is used. Upon checkout, a IntelliJ IDEA project is created around the downloaded sources.

On this page:

– Downloading data to a new workspace generated automatically
– Downloading data to an existing workspace

To have a new workspace generated

1. On the main menu, choose VCS | Checkout from Version Control | TFS. The Checkout from TFS wizard starts.
2. On the Checkout Mode page, choose the Create workspace automatically option, specify the name to identify the workspace, and click Next.
3. On the Source Server page, specify the server to download the data from. Do one of the following:
   – Choose one of the existing server access configurations.
   – Click Add and define a new server access configuration in the Add Team Foundation Server dialog box, that opens.

   Click Next, when ready.
4. On the Choose Source and Destination Paths page, define the working folder mapping to generate a workspace around:
   1. In the Source path area, select the remote folder to download.
   2. In the Destination path text box, specify the location to store the downloaded data in. Type the path manually or click the Browse button … and select the folder in the dialog box, that opens.

   Click Next.
5. On the Summary page, check the details of the workspace to be generated and click Finish to launch the checkout procedure. When the checkout is completed, IntelliJ IDEA creates a project around the downloaded sources and suggests to open it.

To download data to an existing workspace

1. On the main menu, choose VCS | Checkout from Version Control | TFS. The Checkout from TFS wizard starts.
2. On the Checkout Mode page, choose the Choose workspace manually option and click Next.
3. On the Source Workspace page, choose the server to download the data from and the workspace of this server to add the working folders to. If necessary, create new server access configurations and workspaces .
   Click Next, when ready.
4. On the Choose Source Path page, specify the remote folder to download. Such folders are available only if the selected workspace contains a mapping either for the folder itself or for its parent. If such mapping is missing, return to the Source Workspace page and add the required mapping .
   Click Next, when ready.
5. On the Summary page, check the details of the workspace to be generated and click Finish to launch the checkout procedure. When the checkout is completed, IntelliJ IDEA creates a project around the downloaded sources and suggests to open it.

TFS Check-in Policies

A check-in policy is a rule that is executed before every check-in to ensure that the selected changeset is OK to commit. Standard policies are stored on the server and are executed on the client machines.

Custom policies are implemented as custom plugins to IntelliJ IDEA. The IDs of these plugins are stored on the server, while the policies themselves are applied locally. Therefore, to enable the use of a policy in a team, all the team members should install the corresponding plugin.

On this page:

- Defining the default policy settings
- Suppressing the default IntelliJ IDEA-wide check-in policy settings in a current project
- Managing the list of available policies
- Introducing a custom check-in policy

To define the default policy settings to be applied at the IntelliJ IDEA level

1. Open the Settings dialog box , and then click TFS under the Version Control node.
2. On the TFS page, that opens, select the applicable checkboxes in the Checkin policies compatibility area.
   - Evaluate Team Explorer policies: select this check box to have the Microsoft Team Explorer policy definitions installed and executed on the client machine.
   - Evaluate Teamprise policies: select this check box to have the Teamprise policy definitions installed and executed on the client machine.
   - Warn about not installed policies: select this check box to have warnings displayed in case the specified policy definition is not installed.

To suppress applying the default check-in policy settings to a project

1. Open the Settings dialog box , and then click TFS under the Version Control node.
2. On the TFS page, that opens, click the Manage button in the Servers and Workspaces area.
3. In the Manage TFS Servers and Workspaces dialog box, that opens, select the project in question from the Team project drop-down list.
4. In the Compatibility area, select the Override default settings for team project <project name> check box.
5. Re-define the default settings by selecting or clearing the corresponding check boxes below.
   - Evaluate Team Explorer policies: select this check box to have the Microsoft Team Explorer policy definitions installed and executed on the client machine.
   - Evaluate Teamprise policies: select this check box to have the Teamprise policy definitions installed and executed on the client machine.
   - Warn about not installed policies: select this check box to have warnings displayed in case the specified policy definition is not installed.

To manage the list of available policies

The list of available policies consists of standard third-party policies and custom, user-defined policies.

1. Open the Settings dialog box , and then click TFS under the Version Control node.
2. On the TFS page, that opens, click the Manage button in the Servers and Workspaces area.
3. In the Manage TFS Servers and Workspaces dialog box, that opens, select the required workspace and click the Checkin Policies button.
4. In the Edit Checkin Policies dialog box, that opens, configure the list of policies:
   - To activate a policy, select the Enabled check box next to it.
   - To suppress a policy, clear the Enabled check box next to it.
   - To discard a policy permanently, select it in the list and click the Remove button.

To introduce a custom check-in policy

1. Implement the required policy as a custom plugin .
2. Download, install, and enable the plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Using Visual SourceSafe Integration

The Visual SourceSafe (VSS) integration in IntelliJ IDEA is a helpful tool to work with a Visual SourceSafe repository. IntelliJ IDEA implements the most frequently needed commands (Open Source Safe Explorer, Check In/Out, Add, Undo Checkout, Get Latest Version); Next, Previous, Rollback, Old text actions are available from the dedicated gutter bar in changed locations. Thus, the integration is not a complete substitute for the fully functional VSS client.

To use SourceSafe integration with IntelliJ IDEA, make sure that:

- Visual SourceSafe client is installed.
- The following limitations to the VSS options are observed in the SourceSafe Options dialog box:
  - The option Use read-only flag for the files that are not checked out is checked in the Local Files tab.
  - The Assume working folder based on the current project and Assume project based on working folder options should not be checked in the Command Line tab.

Using GitHub Integration

IntelliJ IDEA supports integration with the GitHub remote storage.

To use GitHub integration, perform these general steps

- Enable the GitHub bundled plugin to get access to GitHub integration.
- Register your GitHub account in IntelliJ IDEA.
- Clone repositories from GitHub.
- Publish your projects on GitHub.
- Share code snippets through Git gists.

Registering GitHub Account in IntelliJ IDEA

To retrieve data from GitHub repositories and share your projects in right from IntelliJ IDEA, you need to register your GitHub account credentials in the IDE. You can also create an account on the GitHub free hosting without leaving IntelliJ IDEA.

In either case, IntelliJ IDEA remembers the login and password so you do not need to specify them while retrieving or uploading data.

To register GitHub account credentials

1. Open the IDE Settings and click GitHub.
2. On the GitHub page that opens, select the type of authentication that you want to use from the Auth Type drop-down list. The following options are available:
   - Password. If this option is selected and you have two-factor authentication enabled in your GitHub account settings, you will be asked to enter an authentication code each time IntelliJ IDEA requires you to log in to your GitHub account.
   - Token (recommended by GitHub for authentication from third-party applications, as it does not require IntelliJ IDEA to remember your password).
3. Specify the credentials depending on the selected authentication type and click OK.

To create a GitHub account from IntelliJ IDEA

1. Open the IDE Settings and click GitHub.
2. On the GitHub page that opens, click the Sign up link.
3. On the Sign up for GitHub page that opens in the browser, choose the account plan (free or paid) and specify the requested information. When you are through with creating the account, the GitHub Welcome Page is displayed.
   Free accounts do not support private repositories. Private repositories are available only for you, for other users even READ access is prohibited.
4. Return to the GitHub page and specify your GitHub account credentials in the Login and Password text boxes and click OK.

Cloning a Repository from GitHub

Besides the standard procedure for cloning a remote repository, IntelliJ IDEA provides facilities to clone a repository located on the GitHub remote storage.

Tip Make sure, you have registered your GitHub account credentials in IntelliJ IDEA.

To clone a repository from the GitHub storage

1. Choose VCS | Checkout from Version Control | GitHub on the main menu. IntelliJ IDEA establishes connection with GitHub using the login and password you registered . Upon establishing connection, the Select Git Hub Repository to Clone dialog box opens.
2. From the Repository drop-down list, select the source repository to clone the data from.
3. To view the details and description of the selected repository in the browser, click the Details on <repository name> here link.
4. In the Folder text box, specify the directory where the local repository for cloned sources will be set up. Type the path to the directory manually or click the Browse button ... and choose the desired directory in the Select project destination folder dialog box that opens.
5. In the Project name text box, specify the name of the project to be created based on the cloned sources.
6. Click the Clone button to start cloning the sources from the specified remote repository.

Viewing the GitHub Version of a File

You can view the GitHub copy of a file right from the IDE. IntelliJ IDEA detects which branch is currently active and opens the GitHub copy of the file in the corresponding branch.

To view the remote copy of a file

1. Open the file in the editor or select it in the Project tool window.
2. On the context menu, choose Open in Browser. The GitHub copy of the file in the remote version of the current branch opens.

Note If the command is invoked on a folder, the page with the information on the corresponding remote folder is displayed.

Publishing a Project on GitHub

You can publish your project sources in a GitHub Repository right from IntelliJ IDEA.

Tip Make sure that you have registered your GitHub account credentials in IntelliJ IDEA.

To publish your project sources on GitHub

1. On the main menu, choose VCS | Import into Version Control | Share Project on GitHub.
   - If you have registered login and password , IntelliJ IDEA establishes connection with GitHub using these credentials.
   - If you have not registered your GitHub credentials in IntelliJ IDEA, the Login to GitHub Dialog opens. Specify your GitHub login and password or create an account there.

2. Upon establishing connection, the Share Project on GitHub dialog box opens. Specify the name of the repository to store your project sources in. By default, IntelliJ IDEA suggests the name of the current project. Provide a brief description of the project functionality.
   Click the Share button. IntelliJ IDEA initiates creation of the new repository on the GitHub and uploads the project sources to it.

Creating Gists

To share your code, you can create gists right from IntelliJ IDEA. A gist can contain a code snippet, an entire file, several files, and even folders.

You can also use gists to save and share console output during running, debugging, testing, etc.

On this page:

- Enabling creation of gists
- Saving code in a gist
- Saving console output in a gist

Enabling creation of gists

1. Make sure the GitHib plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .
2. Create an account on the GitHub remote storage, and register this GitHub account in IntelliJ IDEA.

Saving code in a gist

1. Select the code to be shared by doing one of the following:
   - To have a piece of code from a single file shared, open the file in the editor and select the required code snippet. If no selection is made, the gist will contain the entire file.
   - To have the code from a file shared, select the file name in the Project view.
   - To create a gist from a folder, select the folder in the Project view.
   - To create a gist from several files or folders, select them in the Project view.
2. On the context menu of the selection, choose Create gist.
3. In the Create Gist dialog box, that opens, provide a brief description of the gist to be created.
4. Configure the gist's visibility.
   - By default, the new gist will be public, that is, visible for all registered users with your login displayed as owner. To accept this default behaviour, just click OK.
   - To create a private gist that will be available for you only, select the Private check box.
   - To make the gist available for all registered users without displaying your login, select the Anonymous check box.
5. Specify whether you want IntelliJ IDEA to open the new gist in a browser or not:
   - To have the gist opened in the default IntelliJ IDEA browser so you can edit it right now, select the Open in browser check box. When you click OK, the https://gist.github.com/ page, that opens, shows the new gist with the selection or the entire contents of the current file. By default, the gist is named after the file the gist originates from. To update the code snippet, if necessary, click Edit.
   - If you do not want to open the gist right now, clear the Open in browser check box. IntelliJ IDEA displays the URL address of the gist so you you can access it later.

Saving console output in a gist

1. During a run, debug, or test session, or when running commands in the command line mode, switch to the relevant tool window or tab that presents the output of the application or command.
2. Click the right mouse button anywhere in the tab and choose Create Gist on the context menu.
3. In the Create Gist dialog box, that opens, provide a brief description of the gist to be created and configure the gist's visibility .
4. Specify whether you want IntelliJ IDEA to open the new gist in a browser or not .

Creating a Pull Request

Creating pull requests is a nice way to share your code with the community and to follow the collaborative development. With it you can tell others about the changes you've made and ask for comments, review, or just share the knowledge. With all this going on, a pull request appears in the original repository only after approval. A pull request can be prepared right from IntelliJ IDEA without switching to the browser.

To create a pull request:

1. On the main menu, choose VCS | Git | Create Pull Request. The Create Pull Request dialog box opens.
2. In the Base Form field, specify the repository to apply the changes to. IntelliJ IDEA attempts to retrieve all the relevant repositories and shows them in the drop-down list. Choose the repository from the drop-down list or click Select Other Fork and from the Form Owner drop-down list choose the name of the user who is the owner of the target repository.
3. In the Base Branch field, specify the branch to apply the changes to. Click Show Diff to view the list of commits to be included in the pull request. To view the details of a commit, select it and switch to the Log tab which shows a list of files included in the selected commit list. See also Merging, Deleting, and Comparing Branches .
4. In the Title field, specify the name for your request.
5. In the Description field, optionally provide a brief description of the changes to be applied through the request.

Managing Plugins

In this section:

- Plugins
- Enabling and Disabling Plugins
- Installing, Updating and Uninstalling Repository Plugins
- Managing Enterprise Plugin Repositories
- Installing Plugin from Disk
- Adding Plugins to Enterprise Repositories

Plugins

On this page:

- Basics
- Categories of plugins
- Plugin repositories
- Plugin development

Basics

Plugins are extensions to IntelliJ IDEA core functionality. They provide the IDE integration with version control systems (VCS) and application servers, add support for various development technologies, frameworks and programming languages, and so on.

The more plugins are installed and enabled, the more features you have available. On the other hand, disabling unnecessary plugins may increase the IDE performance, especially on "less powerful" computers.

Certain plugins are independent, certain aren't. Dependent plugins require other plugins to be enabled.

Categories of plugins

In relation to IntelliJ IDEA, plugins may be attributed to one of the following categories:

- Plugins bundled with the IDE. These plugins are installed and enabled by default. You can disable unnecessary bundled plugins, but you cannot uninstall them. See Enabling and Disabling Plugins .
- Repository plugins, that is, plugins stored in plugin repositories (e.g., the JetBrains Plugin Repository). To be able to use the repository plugins, you should download and install them. See Installing, Updating and Uninstalling Repository Plugins .

Plugin repositories

IntelliJ IDEA provides access to the JetBrains Plugin Repository at http://plugins.jetbrains.com and to the dedicated IntelliJ IDEA Plugin Repository. The IntelliJ IDEA Plugin Repository resides at http://plugins.jetbrains.com/?idea and stores numerous plugins created by the IntelliJ IDEA community members.

You can set up your own, enterprise plugin repositories, for example, to store plugins that you want to reserve for your company's internal use only. (A plugin repository corresponds to one or more Web servers.)

You can register such repositories in IntelliJ IDEA and then work with them the same way as with the JetBrains Plugin Repository. See Managing Enterprise Plugin Repositories and Adding Plugins to Enterprise Repositories .

Plugin development

IntelliJ IDEA provides an open API that enables you to extend the IntelliJ IDEA functionality: add new intention actions , code inspections and refactorings, facilities for integrating the IDE with various external systems, and more.

For plugin development, IntelliJ IDEA provides dedicated SDK , module and run/debug configuration types.

For more information, refer to:

- Plugin Development Guidelines .
- Information for Plugin Developers .

Enabling and Disabling Plugins

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, select Plugins.
3. On the Plugins page that opens in the right-hand part of the dialog, do one of the following:
   - To enable a plugin, select the check box to the right of its name.
   - To disable a plugin, clear the corresponding check box.
4. If the plugin of interest is not present in the list (which may be the case for a repository plugin), download and install the plugin.
5. Restart IntelliJ IDEA for the changes to take effect.

Installing, Updating and Uninstalling Repository Plugins

On this page:

- Introduction
- Downloading and installing repository plugins
- Updating repository plugins
- Uninstalling repository plugins

Introduction

To be able to use repository plugins , you have to download and install such plugins first. After that, you get the ability to update these plugins when appropriate.

When a repository plugin becomes unnecessary, you can disable or uninstall it.

Downloading and installing repository plugins

To download and install a repository plugin

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, select Plugins.
3. On the Plugins page that opens in the right-hand part of the dialog, click the Install JetBrains plugin or the Browse repositories button.
4. In the dialog that opens (the Browse JetBrains Plugins Dialog or the Browse Repositories dialog), right-click the required plugin and select Download and Install.
   Note that when looking for the plugin of interest, you can filter the plugin list and also to perform a search.
5. Confirm your intention to download and install the selected plugin.
6. Click Close.
7. Click OK in the Settings dialog and restart IntelliJ IDEA for the changes to take effect.

Note that the plugin you have installed is automatically enabled. When necessary, you can disable it as described in Enabling and Disabling Plugins .

Updating repository plugins

To update a repository plugin

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, select Plugins.
   The Plugins page opens in the right-hand part of the dialog.
3. Right-click any of the plugins and select Reload List of Plugins.
   The names of repository plugins that have newer versions available are shown blue.
4. Right-click the necessary plugin and select Update Plugin.
5. Click OK in the Settings dialog and restart IntelliJ IDEA for the changes to take effect.

Uninstalling repository plugins

To uninstall a repository plugin

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, select Plugins.
3. On the Plugins page that opens in the right-hand part of the dialog, right-click the repository plugin to be uninstalled and select Uninstall.
4. Confirm your intention to uninstall the selected plugin.
5. Click OK in the Settings dialog and restart IntelliJ IDEA for the changes to take effect.

Managing Enterprise Plugin Repositories

To be able to use plugin repositories other than the JetBrains Plugin Repository (e.g., your enterprise plugin repositories), you should specify the URLs of such repositories in IntelliJ IDEA.

To manage the list of enterprise plugin repositories

1. In the Settings dialog, click Plugins .
2. Click Browse repositories.
3. In the Browse Repositories dialog that opens, click Manage repositories.
4. Use the Custom Plugin Repositories dialog that opens, to manage the list of URLs for custom (enterprise) plugin repositories:
   - To add a repository URL, click + (`Alt+Insert`). In the Add Repository dialog, specify the repository URL and click OK. (You can use the Check Now button to make sure that the specified URL is correct: IntelliJ IDEA will try to connect to the repository.)
   - To edit a repository URL, select the URL and click (`Enter`). In the Edit Repository dialog, edit the URL and click OK.
   - To remove a URL from the list, select the URL and click – (`Alt+Delete`).

   Click OK in the Custom Plugin Repositories dialog.
5. Click Close in the Browse Repositories dialog.
6. Click OK in the Settings dialog.

As an alternative, you can specify the list of URLs for the enterprise plugin repositories in one of the following files:

- `idea.properties`
  In this case, the following line should be added to this file:

```
idea.plugin.hosts=[URL1];[URL2];...[URLn]
```

- `IntelliJ IDEA.exe.vmoptions`
  In this case, the following line should be added to this file:

```
-Didea.plugin.hosts=[URL1];[URL2];...[URLn]
```

Both files reside in the `bin` directory under the IntelliJ IDEA installation folder.

Note that a repository added via this property is not visible in the Manage Repositories dialog.

Installing Plugin from Disk

Warning! Before installation, make sure that the desired plugin has already been downloaded and saved on your computer as `.zip` or `.jar` file.

To install a plugin from the disk

1. Open the Settings/Preferences dialog box and select Plugins on the left pane.
2. On the right pane of the dialog, click the Install plugin from disk button.
3. In the dialog that opens, select the desired plugin. You can quickly locate and select the necessary file if you drag-and-drop the corresponding item from your file browser (Explorer, Finder, etc.) into the area where the tree is shown. Click OK to proceed.
4. Click Apply button of the Settings/Preferences dialog.
5. Following the system prompt that appears, restart IntelliJ IDEA to activate the installed plugin, or postpone it, at your choice.

Adding Plugins to Enterprise Repositories

To be available to IntelliJ IDEA users, all enterprise repository plugins must be listed in the file `updatePlugins.xml` along with their URLs and version numbers. This file must be available at the URL specified for the corresponding repository (see Managing Enterprise Plugin Repositories).

The plugins themselves are identified by their individual URLs and thus may be located on different Web servers.

– Adding a plugin to an enterprise plugin repository

– DTD for updatePlugins.xml

To add a plugin to an enterprise plugin repository

1. Upload your plugin JAR onto a Web server.
2. Add the plugin definition to `updatePlugins.xml` . If this file doesn't yet exist, create it at the location corresponding to the repository URL.
   The plugin definition in `updatePlugins.xml` may look similar to this:

```
<plugins>
                    ...
                    <plugin id="MyPlugin" url="http://plugins.example.com:8080/myPlugin.jar" version="1.0"/>
                    ...
                    </plugins>
```

3. To publish a new version of the same plugin, upload the corresponding plugin JAR to the repository, and change the value of the `version` attribute in the plugin definition.

DTD for updatePlugins.xml

The file `updatePlugins.xml` must correspond to the following Document Type Definition (DTD):

```
<!DOCTYPE plugins [
                        <!ELEMENT
                          plugins(plugin)*>
                    <!ELEMENT
                          plugin (#PCDATA)>
                    <!ATTLIST
                          plugin id CDATA #REQUIRED url DATA #REQUIRED version CDATA #REQUIRED>]>
```

Managing Tasks and Context

In this section:

- Managing Tasks and Context
  - Basics
  - Prerequisites
  - Supported issue tracking systems
- Enabling Integration with an Issue Tracking System
- Opening and Creating Tasks
- Deleting Tasks
- Switching Between Tasks
- Viewing Description of a Task
- Saving and Clearing Contexts
- Switching Between Contexts
- Tutorial: Configuring Generic Task Server

Basics

IntelliJ IDEA provides facilities to set up your workflow according to the issue tracking procedure accepted in your team.

You can bind your account in an issue tracker to your project, and work on it in the discourse of tasks and contexts.

- A task is an activity performed in IntelliJ IDEA. Each task is identified by a task name.
  Normally, a task correlates with an issue in your issue tracking system. This correlation is set by using the desired issue ID as the task name. When you switch between tasks, IntelliJ IDEA cleans your workspace, creates a changelist for the task, and loads a stack trace, if any.

  Alternatively, you can define a task yourself so it reflects an activity that is not registered in your issue tracker.

  Tasks are listed in the drop-down list on the toolbar. This drop-down list is available only when you have at least one opened issue from the tracker or a self-defined task. A tracker issue has the light-violet background colour until it is opened in IntelliJ IDEA. After a tracker issue is opened in IntelliJ IDEA its background colour changes to white. Self-defined tasks always have the white background colour.
- A context is a set of files opened in the editor while working on a task or independently from it. You can switch between contexts by switching between tasks associated with them. Alternatively, you can save and clear contexts independently from any tasks.

Prerequisites

Before you start working with tasks and contexts, make sure that the Task Management plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Supported issue tracking systems

Currently IntelliJ IDEA supports integration with the following issue tracking systems:

- Jira
- YouTrack
- Lighthouse
- PivotalTracker
- Redmine
- Trac
- FogBugz
- Mantis
- Generic server
- Assembla
- Sprint.ly
- GitHub

Enabling Integration with an Issue Tracking System

On this page:

- Introduction
- Enabling issue tracker's integration

Introduction

To access your issues in the tracking system right from IntelliJ IDEA, you need to integrate your issue tracker with IntelliJ IDEA. Once opened in IntelliJ IDEA, an issue is considered a newly created IntelliJ IDEA task and added to the drop-down list on the toolbar, whereupon you can switch to it from another task. In the terms of IntelliJ IDEA task, opening an issue is referred to as creating a task .

Enabling integration between IntelliJ IDEA and an issue tracking system allows you to work on your project in the discourse of tasks and contexts and thus set up your workflow in accordance with the process established in your team.

The Tasks drop-down list is available only when you have at least one opened issue from the tracker or a self-defined task.

Enabling issue tracker's integration

To enable integration with an issue tracking system

1. Access the Servers dialog box. Do one of the following:
   - Open the Settings dialog box . Below the Tasks node, click Servers.
   - In the Open Task dialog box, click .
   - On the main menu, choose Tools | Tasks&Contexts | Configure Servers

2. In the Servers dialog box, specify the following:
   - The URL address of your issue tracking server.
   - Your account credentials on the server in question. These credentials will be different for the different issue tracking systems.
   - Specify whether you want to access the server via proxy and specify the proxy settings.
   - To allow access to the specified server for other members of your team, select the Share URL check box.
   - To check whether the specified settings ensure successful connection to the server, click the Test button.

   See reference page for the detailed description of controls.

3. Configure synchronization between IntelliJ IDEA and your issue tracking system. To do so, Open the Settings dialog box , and click Tasks. In the Tasks page, configure interaction between IntelliJ IDEA and your tracker. Do one of the following:
   - To have IntelliJ IDEA synchronize with the issue tracking system in the background on a regular basis, select the Enable issue cache check box and specify the synchronization frequency and the cache size.
     No matter whether you actually request on information from your issue tracker or not, IntelliJ IDEA will connect to your issue tracking system according to the specified frequency and refresh the cached issues. The advantage of this approach is that when you need to switch to a task, the up-to-date information is already at your disposal so you do not need to wait till IntelliJ IDEA establishes connection with the tracker and retrieves the information.

     Tip This configuration is especially recommended when working with rather "slow" issue tracking systems.
   - To have IntelliJ IDEA connect to the issue tracking system only when you actually need information on issues, clear the Enable issue cache check box.

Note If a server is not trusted, IntelliJ IDEA shows a dialog box suggesting you to accept the server, or reject it. If you accept the server as trusted, IntelliJ IDEA writes its certificate to the trust store. On the next connect to the server, this dialog box will not be shown.

Opening and Creating Tasks

On this page:

- Basics
- Opening tracker tasks and creating local tasks
- Tips and Tricks

Basics

IntelliJ IDEA distinguishes among **tracker tasks** and **local tasks**.

A **tracker task** is an issue in your tracker system until it is opened in IntelliJ IDEA whereupon it becomes a **local task**.

In addition to this, any activity in IntelliJ IDEA can be configured as a **local task**. This action is referred to as **task creation**.

Opening tracker tasks and creating local tasks

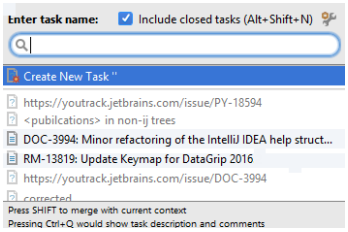
When you open a **tracker task** (an issue in your tracking system) for the first time, a corresponding **local task** is created.

To open tracker tasks and create local tasks, follow these steps

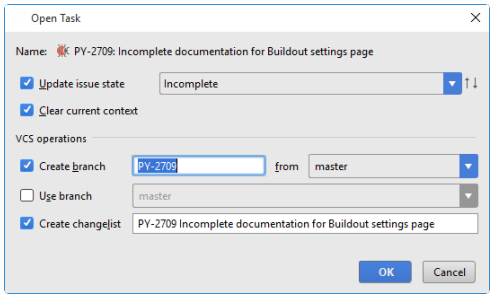
1. Do one of the following:
   - On the main menu, choose Tools | Tasks&Contexts | Open Task.
   - Press `Shift+Alt+N`
   - Click the tasks combo on the main toolbar.

2. To have the possibility to open already closed tasks, select the Include closed tasks check box. A **closed task** is one of the following:
   - A **tracker task** with the status **closed** in the issue tracker.
   - A **local task** that is not associated with a changelist provided that the entire project or the affected directory is under version control (see Associating a Project Root with a Version Control System and Associating a Directory with a Specific Version Control System).

3. In the Enter task name pop-up window, choose Create New Task, or just type the task name.



   - If you choose Create new task <description>, a new local task with the specified description will be created. In this case, the IDE will switch to this new task.
   - If you select a task from an issue tracker, then the Open Task dialog box opens.



   In this dialog, you can specify whether you want IntelliJ IDEA to clear the current context, create a changelist for the new task, change the issue status in the tracking system, create a new VCS branch or use the existing one. Refer to the dialog description for details.

Tips and Tricks

- To show task description right in the suggestion list, press `Ctrl+Q`.
- To configure access to your issue tracking system, click , and specify your account settings in the Servers dialog box that opens.
- To include closed tasks in the suggestion list, press `Shift+Alt+N` once more, or select the corresponding check box.

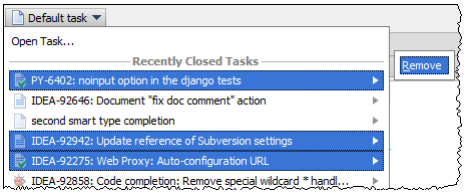
Deleting Tasks

To delete a task

1. Click the tasks combo on the main toolbar.

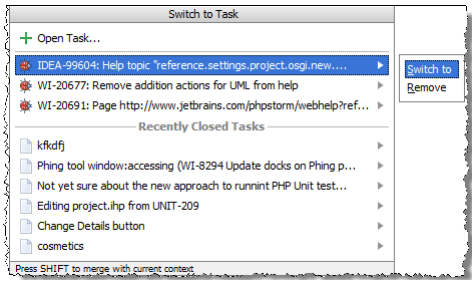
2. Select one or more tasks you want to delete.
   Use `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys for multiple selection.

3. Click the right-arrow button, and choose Remove from the list:

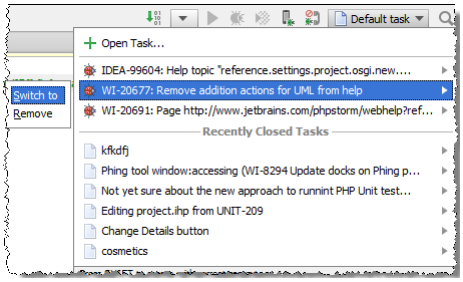


To switch to another task, do one of the following

- Choose Tools | Tasks & Contexts | Switch Task on the main menu. Then in the Switch to Task pop-up window, select the desired task from the list.



- Click the tasks combo in the main toolbar, select the desired task from the list, and then click Switch to on the submenu.



Viewing Description of a Task

On this page:

– Introduction
– Viewing task descriptions
– Opening tasks in browser

Introduction

When you are choosing a task to switch to , the list in the Switch to task pop-up window shows only IDs of tasks and their summaries. This information is not always sufficient, because it reflects neither the steps that lead to the problem nor the related discussion.

You can learn more about your current task in two ways:

– View the description of a task in a pop-up window.
– Open a task in the browser and view both the issue description and all the comments on it.

Viewing task descriptions

To view the description of the current task in the pop-up window

– Choose Tools | Tasks&Contexts | Show '<task ID>' Description on the main menu.

Opening tasks in browser

To open the current task in browser

– Choose Tools | Tasks&Contexts | Open '<task ID>' in Browser on the main menu.

Saving and Clearing Contexts

With IntelliJ IDEA, you can save and clear contexts without associating them with specific tasks .

To manage the current context independently from a task, do one of the following:

- To save the current context, choose Tools | Tasks&Contexts | Save Context on the main menu.
- To clear the current context without loading another one, choose Tools | Tasks&Contexts | Clear Context on the main menu .

Switching Between Contexts

With IntelliJ IDEA, you can switch between contexts that are not associated with specific tasks .

To switch to another context

1. Choose Tools | Tasks&Contexts | Load Context on the main menu or press `Shift+Alt+L`.
2. In the Load Context pop-up window, select the desired context from the list.

Tutorial: Configuring Generic Task Server

On this page:

- Introduction
- Prerequisites
- Specifying the task server
- Configuring the task server
- Choosing the response type and using selectors
- Obtaining issues from the server
- Summary

Introduction

The Generic connector has been developed as a mechanism enabling the users to configure integration with the servers that IntelliJ IDEA does not officially support. At the moment, this mechanism:

- Supports the services with REST API.
- Does not support pagination in server responses.
- Allows the following ways of client authentication:
  - Basic HTTP authentication
  - Sending a preliminary request to the specified address (which is required, for example, by YouTrack).
- Supports GET and POST requests. At this moment POST requests can contain only form-encoded data , specified as query parameters in the corresponding URL

Note that a client can only send requests to a server and retrieve responses according to a certain template.

This tutorial is created for the JIRA 5.0 server. However, configuring a generic server described below is valid for any higher version of JIRA, including JIRA OnDemand.

Prerequisites

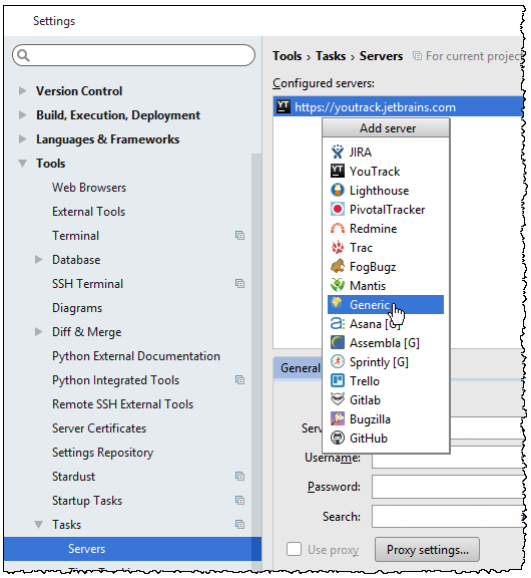
Make sure that the following prerequisites are met:

- You are working with IntelliJ IDEA version 14 or later.
  If you still do not have IntelliJ IDEA, download it from this page . To install IntelliJ IDEA, follow the instructions , depending on your platform.

  This tutorial is created with IntelliJ IDEA version 2016.2.
- You have an account in a tracking system. For example, this tutorial makes use of JIRA.

Specifying the task server

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Tools node, and then click Servers under Tasks.
2. On the Servers page that opens, click + and choose Generic:



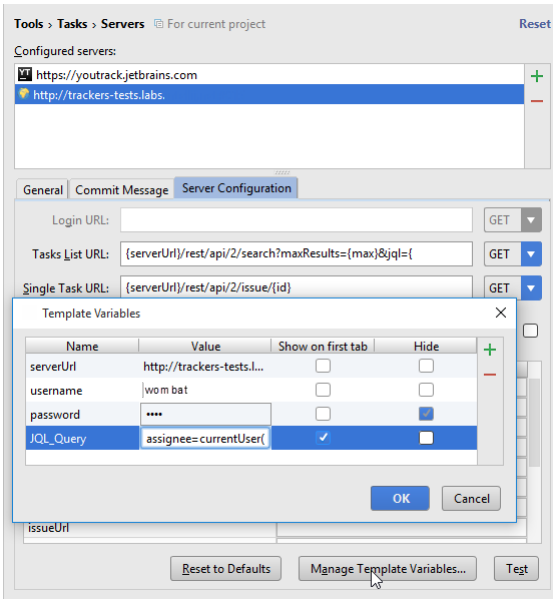
3. In the General tab, specify URL of the server, select the check box Use HTTP Authentication, and type your user name and password.
4. Leave the Commit Message tab as is, and then click the Server Configuration tab.

Configuring the task server

The Server Configuration tab contains three fields:

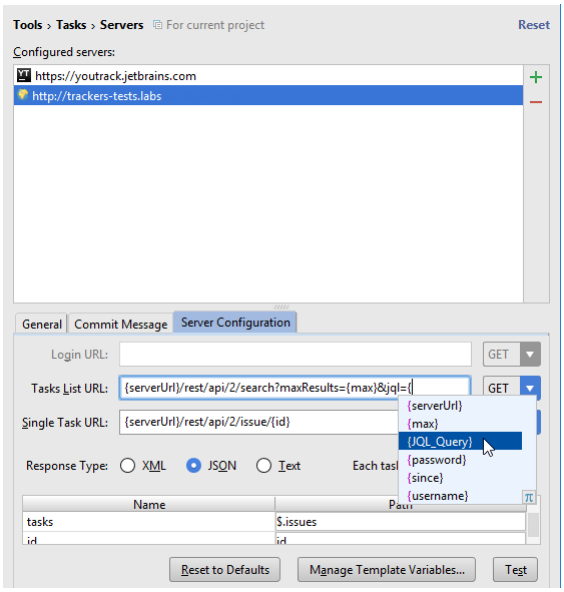
- The Login URL field is disabled, because the check box Use HTTP authentication in the General tab has been selected.
- The Tasks list URL field makes it possible to obtain the list of issues from the server (this list is asked for when you open a task).
  In this field, specify the variable `{serverUrl}` that corresponds to the server URL, specified in the General tab, then type the path `/rest/api/2/search` and expression `?maxResults={max}&jql={JQL_Query}` .

  Pay attention to the template variable `{JQL_Query}` , which is not defined so far. To define it, click the button Manage Template Variables, then in the Template Variables dialog box that opens, click + and type the name and value of this template variable:



Warning! This is important! For an undefined template variable, the test connection will fail!

Defining a template variable leads to including it in the suggestion list:



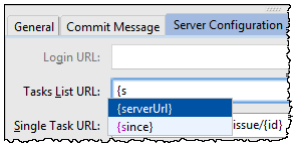
Mind the check box Show on first tab in the Template Variables dialog box. If this check box is selected (as it's done in this tutorial), the template variable and its value shows on the General tab of the Servers page.

Note It is also possible to use an absolute path in the format `http://<server URL>:<port>/rest/api/2/search<expression>` .

- The Single task URL field makes it possible to obtain a detailed information about a specific issue by its ID. This field is optional, when the check box Each task in separate request is cleared.
  To fill in the field, use the following format

  <variable `{serverUrl}` that corresponds to the server URL, specified in the General tab> <the path `/rest/api/2/issue/` > <variable `{id}` >

Code completion is available in all these fields.



Choosing the response type and using selectors

You also need to choose the response type and to specify how information about an issue will be extracted from the server response. This is done in the table of selectors in the lower part of the Server Configuration tab.

IntelliJ IDEA suggests three response types: XML, JSON and text.

- Choose XML if the server responds in the XML format, the selectors are described in XPath.
- Choose JSON if the server responds in the JSON format, the selectors are described in JSONPath .
- Choose Text if the server responds in the plain text format, the selectors are described as regular expressions.

In this tutorial, for example, let's choose JSON and describe the selectors using JSONPath .

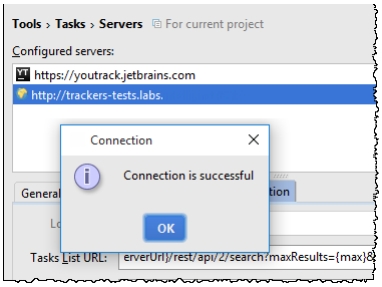
The first three selectors are mandatory:

- tasks: this selector denotes the path in the server response to the descriptions of specific issues.
- id: this selector denotes the path to a unique ID in the description of a specific issue.
- summary: this selector denotes the path to a title of an issue within its description.

To fill out the table, use the server responses. For JIRA, for example, one can observe the server responses immediately in the browser, following a link in the format `http://<server URL>:<port>/rest/api/2/search` .

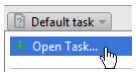
Obtaining issues from the server

1. To check that the specified settings ensure successful integration with the tracking system, click Test:

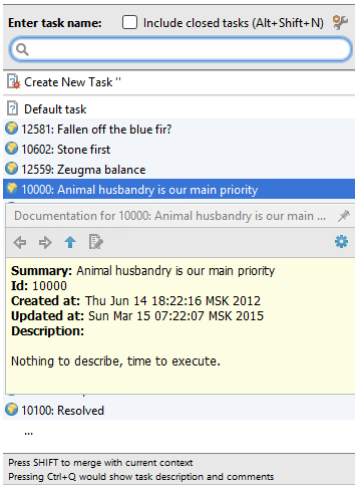


   If the connection is successful, click OK and close the Settings dialog.

2. Do one of the following:
   - On the main menu, choose Tools | Tasks & Contexts | Open Task
   - Press `Shift+Alt+N` .
   - Click the tasks combo on the main toolbar:



3. In the list of tasks, choose the one you want to open, and hit `Enter` :



Summary

In this tutorial, we have:

- assigned JIRA as the generic task server.
- configured this task server.
- specified response type and selectors.
- made sure that connection is successful.
- retrieved issues from the configured server.

Using Command Line Tools

In this section:

- Using Command Line Tools
  - Introduction
  - Prerequisite
- Creating a Command Line Tool
- Running Command Line Tool Commands
- Updating a Command Line Tool

Introduction

You can run command line commands right from IntelliJ IDEA. Just configure the commands that fit your preferences and needs as a command line tool and register it in IntelliJ IDEA. You can have as many command line tools as you need.

Integration with command line tools is provided at the IntelliJ IDEA level, so once configured, a tool can be used in all your IntelliJ IDEA projects. Just activate or de-activate it when necessary depending on the needs of each specific project.

Integration with command line tools is supported via the Command Line Tool Support plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The available tools are listed in the Command Line Tool Support page of the Settings/Preferences dialog.

Prerequisite

Before you start working with Command Line tools, make sure that the Command Line Tool support plugin is downloaded and enabled, as described in Installing, Updating and Uninstalling Plugins .

Creating a Command Line Tool

On this page:

- Introduction
- Creating and configuring a command line tool
- Activating a configured command line tool

Introduction

The command line tools that are available from IntelliJ IDEA are listed in the Command Line Tool Support page of the Settings/Preferences dialog box. Integration with command line tools is provided at the IntelliJ IDEA level, so once configured, a tool can be used in all your IntelliJ IDEA projects. Just activate and de-activate when necessary depending on the needs of each specific project.

You can update the command definitions of a tool and have the `.xml` definition file validated right in the IntelliJ IDEA editor.

Creating and configuring a command line tool

To create and configure a command line tool

1. Open the IntelliJ IDEA settings and click Command Line Tool Support.
2. In the Command Line Tool Support page, click +. In the Command Line Tools dialog box that opens, choose visibility level of the tool (project or global), and its type from the Choose tool drop-down list.
3. In the dialog box that opens, specify tool name and location, and the other required parameters.
   When you click OK, IntelliJ IDEA brings you to the Command Line Tool Support page, where the new tool is added to the list of available tools.
4. Select the newly created tool and click . In the tool definition `.xml` file that opens in the editor, define the tool commands.
5. In the Show console in area, specify where you want the Input pane for typing commands opened:
   - To have the Input pane opened in a pop-up window, choose the Pop-up option.
   - To have the Input pane opened as a text box at the bottom of the Command Line Tools Console tool window, choose the Tool window option.

Activating a configured command line tool

To activate a configured command line tool

1. Open the IntelliJ IDEA settings , and then click Command Line Tool Support.
2. Select the Enable check box next to to the tool name.

Running Command Line Tool Commands

On this page:

- Introduction
- Executing a command in the command line
- Terminating a running command
- Exporting the output
- Launching a command line tool automatically after checking in changes

Introduction

You can run commands and analyze their output right in IntelliJ IDEA using the dedicated Command Line Tools Console tool window.

The character set to be used in the tool window is chosen from the Console encoding drop-down list on the Command Line Tool Support page of the Settings dialog box. If necessary, you can export the output.

Executing a command in the command line

To execute a command in the command line

1. To access the Input pane, choose Tools | Run Command on the main menu.
   Depending on the Show console in setting in the Command Line Tool Support page, the Input pane opens either as a pop-up window or as a text box at the bottom of the dedicated Command Line Tools Console tool window.
2. In the Input pane, type the call of the command in the format:

   ```
   <tool alias>
                        <command>
   ```

   1. Besides tool-specific commands, you can also run simple, operating system-wide, commands, such as, `ls`, etc.
   2. Most command line tools operate within the context of their own projects. A tool project is not the same as a IntelliJ IDEA project. Therefore, when using a tool from IntelliJ IDEA for the first time, you need to create its own "initial" tool project besides you current IntelliJ IDEA project. To do so, run the project generation command specific for the current tool in the Command Line Tool Input pane. The "initial project" is located in the IntelliJ IDEA project root.
3. View the result of command execution in the output tab named after the last invoked command.

Terminating a running command

To terminate a running command, do one of the following:

- During the command execution, click the Stop toolbar button .
- Kill the current thread from the progress bar, if the output tab is already closed.

Exporting the output

To export the output

1. Click the Export to Text toolbar button or press `Alt+O`.
2. In the Export Preview dialog box that opens, do one of the following:
   - To have the output saved in a text file, specify the target text file in the Export to file text box and click the Save button. If the file with this name and location already exists, choose to overwrite its contents with the new data or to append the new data to the existing file.
   - To copy the output to the clipboard, click the Copy button.

Launching a command line tool automatically after checking in changes

To have IntelliJ IDEA launch a command line tool automatically after checking in changes to the repository

1. Open the IntelliJ IDEA settings and configure the required command line tool as an external tool on the External Tools page.
2. Start checking in your changes .
3. In the After Commit area, specify the tool to launch in the Run tool drop-down list. Choose one of the available tools or define a new one: click and configure the required command line tool as an external tool in the External Tools dialog box that opens.

Updating a Command Line Tool

On this page:

– Introduction
– Editing command definitions
– Reloading command definitions
– Validating the definition file for structure consistence

Introduction

You can update the command definitions of a command line tool right in the IntelliJ IDEA editor and have the `.xml` definition file validated . As you type, the file is checked for well-formedness on the fly. Full validation is performed every time you invoke a command. If any inconsistencies are detected, the tool is marked with the Invalid description icon in the Command Line Tool Support page.

You can also add new tasks to the command definition file and enable IntelliJ IDEA to use them without re-configuring the command line tool, that is, without removing it from the list of available ones and adding it anew. All you need, is reload the commands from the executable file.

Editing command definitions

To edit the command definitions

1. Open the IntelliJ IDEA settings , and click the Command Line Tool Support.
2. In the Command Line Tool Support page, select the desired tool and click the Open definition in editor button. The corresponding `.xml` definition file opens in the editor. Update the file as necessary.

Reloading command definitions

To reload command definitions

1. Open the IntelliJ IDEA settings , and click the Command Line Tool Support.
2. In the Command Line Tool Support page, select the desired tool and click the Reload command list from executable button.

Validating the definition file for structure consistence

To validate the definition file for structure consistence

1. Start executing a command : choose Tools | Run Command on the main menu or press `Ctrl+Shift+X`. Then type the call of the command in the Input tab of the Command Line Tools Console tool window.
2. View the results in the Tool definition file errors tab. As soon as IntelliJ IDEA detects any structure inconsistency, it displays an error message that contains a brief description of the problem and indicates the line number where the problem is detected.
   By default, the tab is not displayed and opens only upon validation failure. In this case, IntelliJ IDEA displays a Command Line Tool pop-up window with a notification on validation failure. Upon clicking the More link, the Tool definition file errors tab opens showing messages on detected inconsistencies. Each message contains information on the file and the line number where the problem was found, as well as a brief description of the error.

   You can close the tab by clicking the cross on its header. To re-open the tab, again click More in the Command Line Tool notification pop-up window, which remains on the screen until you close it manually.

Using File Watchers

On this page:

- Basics
- Specifying the File Watcher name, type, and description
- Configuring the behaviour of the File Watcher
- Configuring interaction with the transpiler
- Enabling and disabling File Watchers
- Examples of customizing the behaviour of a transpiler

Basics

IntelliJ IDEA supports integration with various third-party transpilers that run in the background and perform the following:

- Translate Less, Sass, and SCSS source code into CSS code.
- Translate TypeScript and CoffeeScript source code into JavaScript code, possibly also creating source maps to enable debugging.
- Compress JavaScript and CSS code.

Note that IntelliJ IDEA does not contain built-in transpilers but only supports integration with the tools that you have to download and install outside IntelliJ IDEA.

In IntelliJ IDEA, these transpiler configurations are called File Watchers. For each supported transpiler, IntelliJ IDEA provides a predefined File Watcher template. Predefined File Watcher templates are available at the IntelliJ IDEA level. To run a transpiler against your project files you need to create a project-specific File Watcher based on the relevant template, at least, specify the path to the transpiler to use on your machine.

You can download a transpiler of your choice and set it up as a File Watcher. However, in this case no predefined template is available so you will have to specify all the settings manually.

To be applicable, a File Watcher must be enabled by selecting the check box next to it on the File Watchers page of the Settings dialog box, see Enabling and Disabling File Watchers . After that the File Watcher will be invoked automatically upon any changes made to the source code or upon save, depending on whether the Immediate File Synchronization check box is selected or cleared, see New Watcher Dialog .

The output of a File Watcher is stored in a separate file. The predefined templates suggest the type of the file depending on the transpiler type. By default the output file is created in the same folder as the input file when the File Watcher is invoked for the first time, whereupon this file is only updated. You can customize all these settings during File Watcher creation.

JavaScript files generated by File Watchers are excluded from code completion and refactoring.

In the Project tree view, the output file is shown under the original file which is shown as a node. This is done to improve visibility so you can easier locate necessary files.

File watchers have two dedicated code inspections:

- The File watcher available inspection is invoked in every file that is recognized as subject for applying a predefined file watcher (Sass, Less, SCSS, TypeScript, or CoffeeScript). If none of the applicable predefined File Watchers is associated with the current project, IntelliJ IDEA suggests to add one.
- The File watcher problems inspection is invoked by a running File Watcher and highlights errors specific for it.

To create a project File Watcher based on a predefined template, perform these general steps:

1. Make sure the transpiler to use is downloaded and installed on your machine.
2. Make sure the File Watchers repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Specify the File Watcher name, type, and description.
4. In the Options area, configure the behaviour of the File Watcher: the events that will invoke it, from which files it can be invoked, when you want the console shown, etc.
5. In the Watcher Settings area, configure interaction with the transpiler and its behaviour: the path to the executable file, the arguments to pass to it, the acceptable input and expected output file types, etc.

Specifying the File Watcher name, type, and description

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click File Watchers under Tools.
2. The File Watchers page that opens, shows the list of File Watchers that are already configured in the project. Click the Add button + or press `Alt+Insert` and do one of the following:
   - Choose the predefined template to create the File Watcher from. The choice depends on the transpiler you are going to use.
   - To set up a transpiler of your choice, choose Custom.

   After you choose the relevant template or the Custom option, the New Watcher dialog box opens.
3. In the Name text box, type the name of the File Watcher. By default, IntelliJ IDEA suggests the name of the selected predefined template.
4. Provide a brief description of the File Watcher's functionality. Predefined templates already contain general information regarding the accepted input and produced output. Expand or update the description, if, for example, you have two File Watcher that apply to the same source code but differ from each other in some configuration settings. .

Configuring the behaviour of the File Watcher

In the Options area of the New Watcher dialog box, configure the behaviour of the File Watcher: the events that will invoke it, from which files it can be invoked, when you want the console shown, etc.

1. Specify whether you want the File Watcher to interact with the IntelliJ IDEA syntax parser:
   - To have the File Watcher ignore update, save, and change focus events in files that are syntactically invalid and start only in error-free files, select the Trigger watcher regardless of syntax errors check box.

- To have the File Watcher start regardless of the syntactical correctness of a file, clear the Trigger watcher regardless of syntax errors check box. The File Watcher will start upon update, save, or frame deactivation, depending on the status of the Immediate file synchronization check box.

2. Specify the events which will invoke the File Watcher:
   - To have the File Watcher invoked as soon as any changes are made to the source code, select the Immediate file synchronization check box.
   - Clear the check box to have the File Watcher started upon save (File | Save All)or when you move focus from IntelliJ IDEA (upon frame deactivation).

3. Specify how you want the File Watcher to deal with dependencies. When the File Watcher is invoked on a file, IntelliJ IDEA detects all the files in which this file is included. For each of these files, in its turn, IntelliJ IDEA again detects the files into which it is included. This operation is repeated recursively until IntelliJ IDEA reaches the files that are not included anywhere within the specified scope . These files are referred to as root files (do not confuse with content roots).
   - To run the File Watcher only against the root files, select the Track only root files check box.
     Note that in the TypeScript file watcher, the check box has been renamed to Compile main files only.
   - Clear the check box to run the File Watcher against the file from which it is invoked and against all the files in which this file is recursively included within the specified scope.

   This option is available only for Babel, Closure Compiler, Compass, Jade, Less, Sass/ SCSS, Stylus, Typescript, UglifyJS, and YUI Compressor JS.

4. In the Show console drop-down list, specify when you want the console shown. The available options are:
   - Always: when this option is chosen, the File Watcher opens the console when it starts.
   - Error: when this option is chosen, the File Watcher opens the console only if any errors occur in transpilation.
   - Never: choose this option to suppress opening the console under any circumstances.

5. Configure the output filters to distinguish the output of the File Watcher from other output. These filters make the basis for:
   1. Displaying paths to the File Watcher output files as links in error and other messages and logs. When you click such link, the corresponding file is opened in the editor.
   2. Error highlighting in the output files.

   Click the Output Filters button to open the Output Filters dialog and manage the list of filters.

Configuring interaction with the transpiler

In the Watcher Settings area, configure interaction with the transpiler: specify the path to the executable file, the arguments to pass to it, the acceptable input and expected output file types, etc.

1. In the Program text box, specify the location of the transpiler's executable file (`.exe` , `.cmd` , `.bat` or other depending on the specific tool. `.jar` archives are also acceptable but defining `PATH` variables for them is not supported. Do one of the following:
   - Type the path explicitly.
   - Click the Browse button to open the Select Path dialog box and navigate to the desired location.
   - Click the Insert Macro button to open the Macros dialog box where you can select the relevant macro from the list.

2. From the File type drop-down, choose the expected type of input files. The File Watcher will consider only files of this type as subject for analyzing and processing. File types are recognised based on associations between file types and file extensions .
   By default, the field shows the file type in accordance with the chosen predefined File Watcher template.

3. In the Scope drop-down list, define the range of files the File Watcher can be applied to. Changes in these files will invoke the File Watcher either immediately or upon save or frame deactivation, depending on the status of the Immediate file synchronization check box.
   You can choose one of the predefined scopes from the drop-down list:
   - Project Files: all the files within the project content roots (see Content Root and Configuring Content Roots).
   - Project Production Files: all the files within the project content roots excluding test sources.
   - Project Test Files: all the files within the project test source roots.
   - Open Files: all the files that are currently opened in the editor.

   VCS Scopes: these scopes are only available if your project is under version control .
   - Changed Files: all changed files, that is, all files associated with all existing changelists .
   - Default: all the files associated with the changelist `Default` .

   Alternatively, click the Browse button and configure a custom scope in the Scopes dialog box that opens.

   For more details on scopes, see the pages Scopes and Scopes dialog .

   The Scope setting overrides the Track only root files check box setting: if a dependency is outside the specified scope, the File Watcher is not applied to it.

4. In the Arguments text box, define the arguments to pass to the transpiler Among other cases, use this text box to change the default output location, that is, specify a custom location where you want the transpiler to store the files generated during transpilation. Do one of the following:
   - Type the list of arguments in the text box.
   - Click the Insert Macro button to open the Macros dialog box where you can select the desired macro from the list.

   When specifying the arguments, follow these rules:
   - Use spaces to separate individual arguments.
   - If an argument includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` .
   - If an argument includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value"\` .

5. In the Working Directory text box, specify the directory to which the transpiler will be applied. Because the tool is always invoked in the context of a file, the default working directory is the directory of the current file. This setting is specified in all predefined templates through a macros `$FileDir$` . To update this default settings, do one of the following:
   - Type the path explicitly in the text box.
   - Click the Browse button to open the Select Path dialog box and navigate to the desired location.

– Click the Insert Macro button to open the Macros dialog box, where you can select the desired macro from the list.

Tip If the field is left blank, IntelliJ IDEA uses the directory of the file where the File Watcher is invoked.

6. In the Output paths to refresh text box, specify the files where the transpiler stores its output: the resulting source code, source maps, and dependencies. Based on this settings, IntelliJ IDEA detects files generated through transpilation.
   Please note, that changing the value in this text box does not make the transpiler store its output in another location. To do that, specify the desired output location in the Arguments text box.

   Do one of the following:
   – Type the output paths manually. Use colons as separators.
   – Click the Insert Macro button to open the Macros dialog box, where you can select the desired pattern from the list.

7. Use the Create output file from stdout check box to specify how you want the output file generated.
   – To have IntelliJ IDEA read the native transpiler's output (`standard output stream (stdout)`) and generate the resulting files from it, select the Create output file from stdout check box.
   – To have the transpiler write its output directly to the files specified in the Output paths to refresh field, clear the check box.

Tip Some transpilers generate a `standard output stream (stdout)` file, others do not, which may lead to errors. Therefore it is strongly recommended to preserve the default setting.

8. Optionally, define environment variables. For example, specify the `PATH` variable for the tool that is required for starting the transpiler but is not referenced in the path to it. In most cases it is Node.js or ruby.exe. Such situation may happen if you installed the transpiler manually but not through the Node Package Manager or gem manager and its installation folder is not under Node.js or ruby.

Enabling and disabling File Watchers

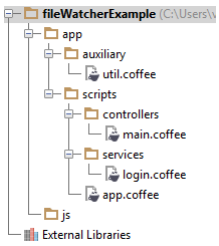
To toggle the enable/disable status of a File Watcher, select/clear the check box next to it on the File Watchers page of the Settings dialog box. If an error occurs while a File Watcher is running, the File Watcher is automatically disabled. To restore the status, enable the File Watcher manually.

When a File Watcher is enabled, it starts automatically as soon as a file to which transpilation is applicable is changed or saved, see Configuring the behaviour of the File Watcher .

Examples of customizing the behaviour of a transpiler

Any transpiler is an external, third-party tool. Therefore the only way to influence a transpiler is pass arguments to it just as if you were working in the command line mode. These arguments are specific for each tool. Below are two examples of customizing the default output location for the CoffeeScript transpiler.

Suppose, you have a project with the following folder structure:



By default, the generated files will be stored in the folder where the original file is. You can change this default location and have the generated files stored in the `js` folder. Moreover, you can have them stored in a flat list or arranged in the folder structure that repeats the original structure under the `app` node.

– To have all the generated files stored in the output `js` folder without retaining the original folder structure under the `app` folder:

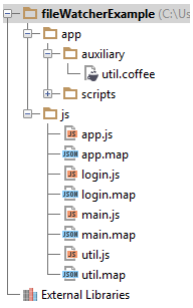
1. In the Arguments text box, type:

```
--output $ProjectFileDir$\js\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir$\js\$FileNameWithoutExtension$.js:$ProjectFileDir$\js\$FileNameWithoutExtension$.map
```

As a result, the project tree looks as follows:



- To have the original folder structure under the `app` node retained in the output `js` folder:

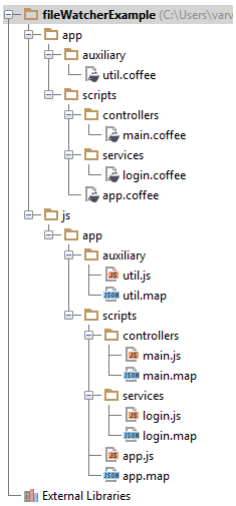
  1. In the Arguments text box, type:

     ```
     --output $ProjectFileDir$\js\$FileDirRelativeToProjectRoot$\ --compile --map $FileName$
     ```

  2. In the Output paths to refresh text box, type:

     ```
     $ProjectFileDir$\js\$FileDirRelativeToProjectRoot$\$FileNameWithoutExtension$.js:$ProjectFileDir$\js\$FileDirRelativeToP
     ```

  As a result, the project tree looks as follows:



Working with Embedded Local Terminal

On this page:

- Prerequisites
- Overview
- Configuring embedded local terminal
- Running embedded local terminal
- Actions available in the embedded local terminal

Prerequisites

Before you start working with terminal, make sure that the Terminal plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Overview

IntelliJ IDEA features a local terminal that makes it possible to access the command line. Depending on your platform, you can work with command line prompt, Far, `powershell` , `bash` , etc. Using the terminal, you can execute any command without leaving the IDE.

Configuring embedded local terminal

The terminal settings are configurable on several pages of the Settings/Preferences dialog.

To configure the embedded local terminal options

1. In the Settings/Preferences dialog box, open the Terminal page, and specify the following:
   - The desired shell that will run by default, the name of a session tab
   - Name of the tab a new session will be opened in, possibility to copy to clipboard etc.
   - Ability to override the IntelliJ IDEA keymap.

2. The following settings are inherited by the embedded local terminal from the IDE settings/preferences:
   - On the Keymap page, you can configure the `Ctrl+C` and `Ctrl+V` shortcuts.
   - On the editor's Appearance page - anti-aliasing and caret blinking.

     Note that the setting Use block caret is not inherited in the terminal - its caret is always block.
   - Under the editor's Color and Fonts settings, you can change the following options:
     - On the Console Fonts page - line spacing, and console fonts.
     - On the Console Colors page - console colors.
     - On the General page - selection foreground and background colors.

Running embedded local terminal

To run the console, do one of the following

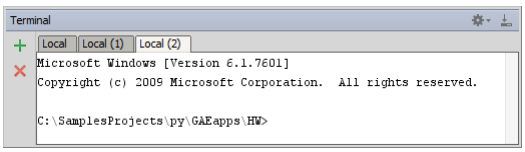
- Press `Alt+F12` .
- Click the Terminal tool window button `Terminal` .
- Hover your mouse pointer over in the lower left corner of the IDE, then choose Terminal from the menu, as described in the section Tool window quick access .

Actions available in the embedded local terminal

In the embedded local terminal, you can do the following:

- To create a new session:
  - Click + on the toolbar of the terminal.
  - Right-click a session tab, and then choose Create new session on the context menu.

  A new session is presented in a separate tab:



- Rename a tab. Right-click a tab, and choose Rename tab on the context menu.
- Close an active session that currently has the focus. This can be done in a number of ways:
  - Click on the terminal toolbar.
  - Right-click a session tab, and then choose Close session on the context menu.
- Use up and down arrows on your keyboard to browse through the history of entered commands.
- Toggle between the embedded local terminal and editor by pressing `Alt+F12` (View | Tool Windows | Terminal).

Comparing Files and Folders

IntelliJ IDEA helps explore differences in the various situations: differences between files, directories, revisions of the same file under version control or in the local history, database objects, local and remote files.

All these operations are performed in a similar way. In this section we'll consider the most basic operations:

- Comparing Files
- Comparing Folders

Comparing Files

On this page:

- Introduction
- Comparing two files
- Comparing a file in the editor with the Clipboard contents
- Comparing a file with the editor contents

Introduction

IntelliJ IDEA enables you to compare arbitrary files in project (including image files), selected file with the editor, or compare a file in the editor with the Clipboard contents.

All comparisons are performed in the Differences viewer .

Comparing two files

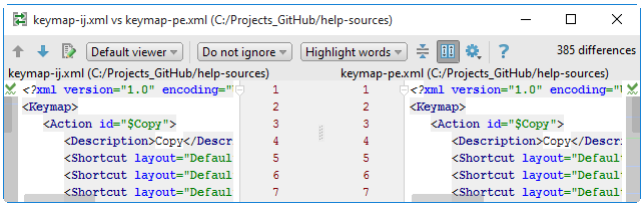
1. Press and keep holding `Ctrl` for Windows and Linux or `⌘` for OS X and click the two files to compare in the Project tool window.
2. On the context menu of the selection, choose Compare Files, or press `Ctrl+D`. The Differences Viewer for Files opens, with the differences being color-highlighted.

   Tip It is enough to select a single file in the Project tool window. In this case the context menu command is Compare File with Editor, and the Differences Viewer shows the contents of the selected file on the left pane, and the contents of the active editor tab on the right pane.

3. View the differences and apply them, if necessary, using the chevron buttons » «.

   Note that keeping `Ctrl` (for Windows and Linux) or `⌘` (for OS X) pressed turns the chevron buttons » « to . Click these buttons to append changes.

   Keeping `Shift` pressed turns the chevron buttons » « to ×. Click this button to revert changes.



Comparing a file in the editor with the Clipboard contents

1. Open the desired file in the editor.
2. Right-click the editor pane and choose Compare with Clipboard on the context menu.
3. View and manage differences in the Differences Viewer for Files .

Comparing a file with the editor contents

1. Right-click the desired file in the Project tool window.
2. Choose Compare File with Editor on the context menu.
3. View and manage differences in the Differences Viewer for Files .

Comparing Folders

On this page:

- Basics
- Opening the Difference Viewer
- Comparing two folders in the Difference Viewer
- Synchronizing contents of folders

Basics

IntelliJ IDEA provides a dedicated Differences Viewer for Folders for comparing files in two folders against the file size, content, or timestamp. The Differences Viewer shows the contents of the selected directories in the left and right panes of the Item List. The contents of the selected file are shown in the lower pane, with the differences being color-highlighted.

Besides exploring differences, the tool also provides interface for synchronizing the contents of folders.

Opening the Difference Viewer

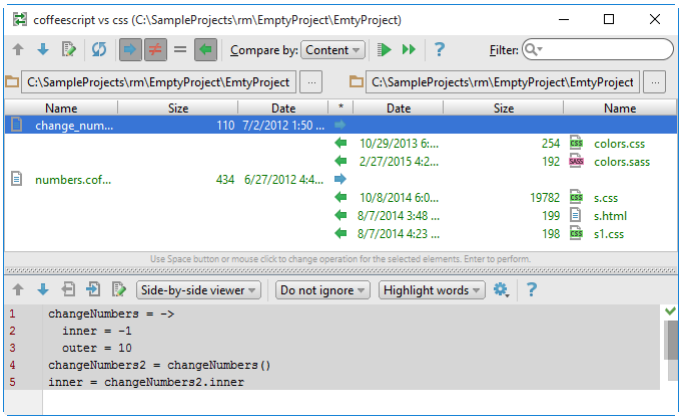
Do one of the following:

- Keeping the `Ctrl` key pressed, click two directories in the Project tool window, and choose Compare Directories on the context menu of the selection, or press `Ctrl+D`.
- Select a directory in the Project tool window, choose Compare with on the context menu of the selection, or press `Ctrl+D`, and then select the second directory in the dialog that opens.

> **Tip** You can also open the difference viewer without running IntelliJ IDEA. This is done through the following command: `<path to IntelliJ IDEA executable file> diff <path_1> <path_2>` where `path_1` and `path_2` are paths to the folders in question.

Comparing two folders in the Difference Viewer

1. Configure the layout of the Items List. Use the toolbar buttons to narrow down or widen the set of items to show. For example, show or hide files that exist in just one of the directories, equal files, or different files, etc.
2. Specify the parameter for comparison. In the Compare by drop-down list, select one of the possible options (contents, size, or time stamp).
3. Filter the folders' contents. To do that, type filtering string in the Filter text field, and press `Enter` to apply it. Using the asterisk `*` wildcard to represent any number of characters is welcome.
4. To switch to another pair of folders to compare, update the fully qualified paths to them. Click the Browse button next to the Paths read-only fields and choose the required folders in the dialog box that opens.
5. Explore the detected differences between files in the Differences Pane.



Synchronizing contents of folders

1. For each pair of items, in the * field specify the action to apply. Click the icon in the field until the required action is set.

Icon	Action
	Copy the item in the left side to the right side, possibly overwriting the contents of the corresponding target item, if it already exists.
	Copy the item in the right side to the left side, possibly overwriting the contents of the corresponding target item, if it already exists.
	The items are treated identical with regard to the selected criterion of comparison. No action will be performed by default.
	The items differ with regard to the selected criterion of comparison. No action will be performed by default. Explore the differences in the Differences Pane and change the intended action by clicking the icon.
	The item is present only in one of the folders and will be removed.

2. Do one of the following:
   - To synchronize the currently selected item, click the Synchronize Selected button on the toolbar.
   - To synchronize all the items, click the Synchronize All button on the toolbar.

Working with Background Tasks

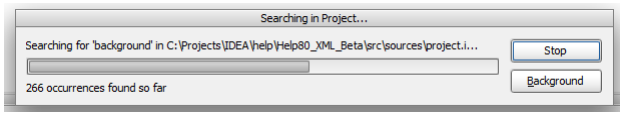
On this page:

– Introduction

– Viewing background tasks

Introduction

When lengthy tasks are running, for example, search and replace, VCS update, etc., IntelliJ IDEA displays the progress bar. You can bring execution of such tasks to the background by clicking the Background button.

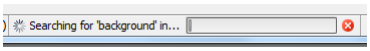


However, when a task of a certain type is brought to the background, you can still view its progress.

Viewing background tasks

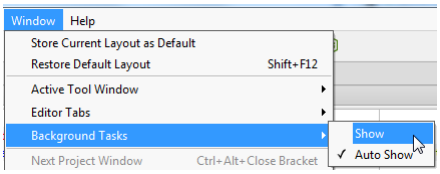
To view tasks running in the background, do one of the following

– View the Status bar:

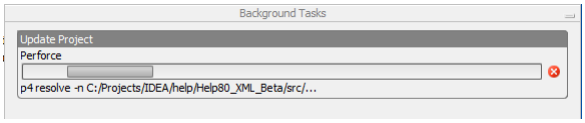


– In the Status bar , click .

– On the main menu, choose Window | Background Tasks | Show.



As a result, the Background Tasks manager opens, showing all the tasks that are currently running in the background. You can review their progress, and if necessary, cancel the unnecessary tasks by clicking .



When the Auto Show command is checked, the Background Tasks manager opens automatically every time a task is launched in the background.

Working with IntelliJ IDEA Features from Command Line

Besides working from within IntelliJ IDEA, it is possible to perform certain actions "offline", without actually launching the IDE. This way you can:

- Launching a code inspection from the command line
  - Examples
    - Windows
    - OS X
- Viewing differences
- Opening a file in the editor

Launching a code inspection from the command line

To launch a code inspection from the command line

- Specify the following command line arguments:
  - Path to the launcher: specify the full path to one of the following launchers (which reside under the `bin` directory of your IntelliJ IDEA installation):
    - For Windows: `inspect.bat`
    - For UNIX and OS X: `inspect.sh`
  - Project file path is the full path to the directory that contains the project to be inspected.
  - Inspection profile path is the full path to the profile, against which the project should be inspected. The inspection profiles are stored under `USER_HOME\.IntelliJ IDEAXX\config\inspection`
  - Output path is the full path to an existing directory where the report will be stored.
  - Options. You can specify:
    - The directory to be inspected `-d <full path to the subdirectory>`
    - The verbosity level of output `-vX`, where X is 0 for quiet, 1 for noisy and 2 for extra noisy.

Warning! Please note that you have to specify full paths. Relative paths are not accepted!

Tip If SDK is not defined, the inspection will fail. The SDK descriptions should be stored in `config\options\jdk.table.xml`. Learn how to configure SDK here.

Examples

Windows

```
C:\Program Files (x86)\JetBrains\<IntelliJ IDEA home>\bin\inspect.bat
E:\SampleProjects\MetersToInchesConverter E:\Work\MyProject\.idea\inspectionProfiles\Project_Default.xml
E:\Work\MyProject\inspection-results-dir -v2 -d E:\SampleProjects\MetersToInchesConverter\subdirectory
```

OS X

```
/Applications/IntelliJ IDEA.app/Contents/bin/inspect.sh ~/IntelliJ IDEAProjects/MyTestProject
~/Library/Preferences/ideaXX/inspection/Default.xml ~/IntelliJ IDEAProjects/MyTestProject/results-dir -v2
```

Viewing differences

To view differences using command line diff tool

- In the command line, type the following:

```
<IntelliJ IDEA> diff <path1> <path2>
```

where:

- <IntelliJ IDEA> is the platform-specific product launcher
- `<path1>, <path2>` are full paths to the files to be compared.

Examples

Windows

```
IntelliJ IDEA.exe diff
C:\SamplesProjects\MetersToInchesConverter\src\javascript\numbers.js
C:\SamplesProjects\MetersToInchesConverter\src\coffeescript\numbers.coffee
```

OS X

```
/Applications/IntelliJ IDEA.app/Contents/MacOS/idea diff ~/Documents/file1.txt ~/Documents/file2.txt
```

Opening a file in the editor

To open a file for editing

- In the command line, type the following:

```
<IntelliJ IDEA> <path1> --line <number> <path2>
```

where:

- `<IntelliJ IDEA>` is the platform-specific product launcher
- `<path1>` is the path to the project that contains the desired file
- `<number>` is the number of the line, where the caret should rest
- `<path2>` is the path to the file to be opened

Examples

Windows

```
IntelliJ IDEA.exe C:\SamplesProjects\MetersToInchesConverter --line 3
C:\SamplesProjects\MetersToInchesConverter\src\javascript\numbers.js
```

OS X

```
/Applications/IntelliJ IDEA.app/Contents/MacOS/idea ~/IntelliJ IDEAProjects/untitled45 --line 1 ~/IntelliJ IDEAProjects/unti
```

Language and Framework - Specific Guidelines

This part part provides descriptions of the actions required to fulfil certain tasks using the frameworks integrated into IntelliJ IDEA:

– ActionScript and Flex
– Android
– Arquillian: a Quick Start Guide
– AspectJ
– Build Tools
– CoffeeScript Support
– ColdFusion
– Context and Dependency Injection (CDI)
– Copyright
– Databases and SQL
– Dart Support
– Docker
– EJB
– Erlang
– Grails
– Griffon
– Groovy
– GWT
– IntelliLang
– Java SE
– Java EE
– JavaFX
– J2ME
– JavaScript-Specific Guidelines
– JavaServer Faces (JSF)
– JPA and Hibernate
– Kotlin
– Markup Languages and Style Sheets
– Node.js
– OSGi and OSMORC
– PHP-Specific Guidelines
– Play Framework 1.X
– Plugin Development Guidelines
– RESTful WebServices
– Scala
– Seam
– Spring
– Struts Framework
– Struts 2
– Swing. Designing GUI
– Tapestry
– Template Languages: Velocity and FreeMarker
– Testing Frameworks
– TextMate
– Thymeleaf
– Tiles 3
– TypeScript Support
– Vaadin
– Vagrant
– Working with Diagrams
– Web Applications
– Web Service Clients
– Web Services
– XML Refactorings
– XPath and XSLT Support
– Tutorial: Using Emacs as an External Editor
– Tutorial: Using IntelliJ IDEA as the Vim Editor

ActionScript and Flex

In this section:

- ActionScript and Flex
  - Basics
  - ActionScript and Flex support
  - FlexUnit support
- Preparing for ActionScript or Flex Application Development
- Creating and Editing ActionScript and Flex Application Elements
- Working with Build Configurations
- Configuring Dependencies for Modular Applications
- Building ActionScript and Flex Applications
- Running and Debugging ActionScript and Flex Applications
- Packaging AIR Applications
- Importing Adobe Flash Builder Projects
- Testing ActionScript and Flex Applications
- ActionScript-Specific Refactorings

Basics

To support ActionScript and Flex, IntelliJ IDEA provides:

- The Flash/Flex Support plugin. This plugin is bundled with the IDE and must be enabled.
- A dedicated module type (Flash).
- Build configurations for the various target platforms (Web, Desktop and Mobile) and output types (SWF and SWC).
- Dedicated run/debug configuration types (Flash App, FlexUnit and Flash Remote Debug).
- The ActionScript Profiler and Flash UI Designer plugins. These plugins are available for download from the JetBrains repository. For more information, see Profiling CPU in Flash and Flex Applications. See also, Installing, Updating and Uninstalling Repository Plugins.

ActionScript and Flex support

ActionScript and Flex support includes:

- Code completion, including completion of statements (`Ctrl+Shift+Enter`) and Smart Type completion (`Ctrl+Shift+Space`).
- Error and syntax highlighting.
- ActionScript and Flex code refactorings:
  - Change Method Signature.
  - Delegate Methods.
  - Extract Interface.
  - Extract Method.
  - Extract Superclass.
  - Inline.
  - Introduce Constant.
  - Introduce Field.
  - Extract Parameter.
  - Introduce Variable.
  - Move a class or an interface to a package.
  - Move Inner to Upper Level for moving classes, functions, variables, constants and namespaces declared outside of packages into a package.
  - Move/Copy a file.
  - Move Static Members.
  - Pull Members up, Push Members down.
  - Rename a file, function, variable, parameter, property or label (both directly and via references).
  - Safe Delete a file.
- ActionScript and Flex code inspections and quick-fixes.
- Intention Actions for creating various application elements.
- Code formatting and folding.
- Advanced Search and Navigation, plus Structure View.
- Enhanced navigation with gutter icons.
- Navigation from CSS properties and selectors to their declarations in ActionScript (`Ctrl+B`).
- Possibility to build ActionScript and Flex applications using various compiler shells and compilation options.
- Support for breakpoints and specific run/debug configurations for debugging ActionScript and Flex applications directly from IntelliJ IDEA.
- ActionScript and Flex-aware debugger that lets you execute applications step by step, evaluate expressions, examine related information and find runtime bugs.
- Quick Javadoc (`Ctrl+Q`) for AsDoc.
- BlazeDS support.
- AIR application development support at all stages of application development cycle. Development of AIR applications for mobile devices is also supported. For basic how-to information, see New in IntelliJ IDEA 10.5: Develop Mobile AIR Applications for Android.
- Possibility to create pure ActionScript applications.
- ActionScript live templates: File | Settings | Live Templates | ActionScript and JavaScript groups.
- Ability to wrap and unwrap code constructs in ActionScript and MXML (`Ctrl+Alt+T` and `Ctrl+Shift+Delete`).
- Type Hierarchy (`Ctrl+H`), Method Hierarchy (`Ctrl+Shift+H`) and Call Hierarchy (`Ctrl+Alt+H`) for Flex sources (`*.mxml` and `*.as` files).

- Easy import of Flex projects created in Adobe Flash Builder .
- Import of Flexmojos projects. See Working with Flexmojos projects in IntelliJ IDEA . See the description of related import setting .
- ActionScript and Flex UML class diagrams . Among the features is the ability to view changes in ActionScript source files in a structured visual form.

FlexUnit support

IntelliJ IDEA supports the versions 0.9 and 4 of FlexUnit , a unit testing framework for Flex and ActionScript applications and libraries.

FlexUnit support includes:

- Dedicated FlexUnit run/debug configurations to run a single test method, test suite, all methods in a certain test class, or all test classes in a given package.
- Ability to perform the tests both in the run and the debug modes.
- Support for Flunit tests via FlexUnit 4 test runner.
- FlexUnit-aware code inspections (turned off by default).

Preparing for ActionScript or Flex Application Development

On this page:

- Preliminary steps
- Registering a Flex or AIR SDK in IntelliJ IDEA
- Configuring general Flex compiler settings
- Creating a Flash module
- Configuring module contents

Preliminary steps

To prepare for ActionScript or Flex application development:

1. Make sure that the Flash/Flex Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .
2. Download and install a Flex or AIR SDK on your computer. IntelliJ IDEA provides integration with Apache Flex , Adobe Flex , Adobe AIR and Adobe Gaming SDKs .
3. Register the Flex or AIR SDK in IntelliJ IDEA. (You can postpone this step until a later time. You will be able to specify the SDK when creating a Flash module .)
4. Check general Flex compiler settings and, if necessary, adjust them to your needs.
5. Create a new project with a Flash module, or add a Flash module to an existing project. See Creating a Flash module .
   Note that you can create a project or a module by importing Adobe Flash Builder projects .
6. Check the initial module configuration and make the necessary adjustments. These may include configuring the module contents , adding libraries , modifying the existing build configuration and creating additional ones, etc.
   All these tasks, however, may be performed at a later time, when needed, in parallel with developing your source code .

Registering a Flex or AIR SDK in IntelliJ IDEA

To register a Flex or AIR SDK in IntelliJ IDEA

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. Under Platform Settings, click SDKs.
3. Click Add New SDK on the toolbar, and then click Flex/AIR SDK in the Add New SDK list.
4. In the dialog that opens , select the directory in which the Flex or AIR SDK is installed and click OK.
   As a result, the SDK configuration is shown on the SDK page in the right-hand part of the Project Structure dialog.
5. Generally, you don't need to make any changes on the Classpath and Sourcepath tabs because all the necessary libraries and sources are already there. However, you may want to add external online documentation to be able to access additional reference information when writing your code. To do that:
   On the Documentation Paths tab, click (`Alt+S`). In the dialog that opens, just click OK. (The URL suggested by IntelliJ IDEA is the one that you want.)
6. Click OK in the Project Structure dialog.

Configuring general Flex compiler settings

To configure general Flex compiler settings

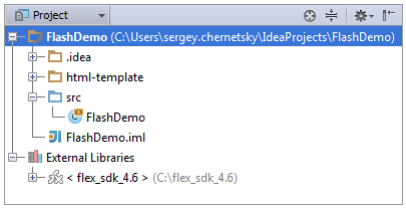
1. Open the Settings dialog (`Ctrl+Alt+S` , File | Settings, or on the toolbar).
2. In the left-hand part of the dialog, under Project Settings, open the Compiler node and click Flex Compiler.
3. On the Compiler > Flex Compiler page , configure the settings as required.

Creating a Flash module

To create a Flash module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Flash .
3. In the right-hand part of the page, select the necessary options. For more information, see Flash.
   Click Next.
4. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

As a result, the module structure looking similar to this is generated (the contents may be different depending on the module options that you have selected):



In this structure:

- `html-template` is a folder with files that constitute an HTML wrapper template.
- `src` is a folder for your application source files (`.as` and, possibly, `.mxml`).

In addition to the module itself, IntelliJ IDEA creates the following:

- One build configuration .
- One run/debug configuration . If when creating the module you have selected to create a sample application, you can use this configuration to run the application straight away. To do that, click on the toolbar (`Shift+F10`).

Configuring module contents

To configure module contents

The module contents are configured by adding and removing the module content roots as well as by assigning individual folders (within the content roots) to source folders, test source folders and also by excluding the folders.

For a Flash module, generally, these task are performed on the Module page of the Project Structure dialog.

To access this page:

1. Open the Project Structure dialog.
2. Under Project Settings, select Modules.
3. In the area under + − | , select the necessary Flash module.

Creating and Editing ActionScript and Flex Application Elements

When working on ActionScript and Flex applications, you create and edit application elements such as packages , ActionScript classes and interfaces , and MXML components .

For ActionScript classes, interfaces and MXML components, IntelliJ IDEA provides a number of predefined file templates .

- Creating a package
- Creating an ActionScript class or interface
- Creating an MXML component
- Initiating ActionScript class creation in the editor
- Template-based ActionScript classes, interfaces and MXML components
- Predefined file template variables for ActionScript and Flex
- An example of creating a custom file template for an MXML component
- Using the SWF metadata tag to control HTML wrapper properties
- Editing ActionScript and Flex sources

Creating a package

1. In the Project tool window, select your source root folder (e.g. `src`) or the package in which you want to create a new package. Choose File | New | Package, or press `Alt+Insert` and select Package.
   Alternatively, right-click the corresponding folder or package and select New | Package from the context menu.

2. In the New Package dialog that opens, specify the package name and click OK.
   Note that you can create more than one package at once if you use dots (`.`) to separate the package names. For example, if you type `myPackage.mySubpackage` and none of these packages currently exists, both these packages (`myPackage` and `mySubpackage`) will be created.

   Also note that you can create a new package when creating a new ActionScript class or interface, or an MXML component. See Creating an ActionScript class or interface and Creating an MXML component.

Creating an ActionScript class or interface

1. In the Project tool window, select your source root folder (e.g. `src`) or the package in which you want to create a new ActionScript class or interface. Choose File | New | ActionScript Class, or press `Alt+Insert` and select ActionScript Class.
   Alternatively, right-click the corresponding folder or package and select New | ActionScript Class from the context menu.

2. In the New ActionScript Class Dialog that opens, specify the name of the class, the package, the file template to be used, and click Create. If the Class with Supers template is used, you can also specify a superclass and/or one or more interfaces that the class should implement.
   Note that if you specify a package that doesn't yet exist, the corresponding package will be created.

See also, Initiating ActionScript class creation in the editor.

Creating an MXML component

1. In the Project tool window, select your source root folder (e.g. `src`) or the package in which you want to create a new MXML component. Choose File | New | MXML Component, or press `Alt+Insert` and select MXML Component.
   Alternatively, right-click the corresponding folder or package and select New | ActionScript Class from the context menu.

2. In the New MXML Component Dialog that opens, specify the name of the component, the package, the file template to be used, the parent component, and click Create.
   Note that if you specify a package that doesn't yet exist, the corresponding package will be created.

Initiating ActionScript class creation in the editor

When working with the source code of a class, you can start creating its subclass right in the editor. Similarly, you can initiate creating a class implementing an interface. For these purposes, IntelliJ IDEA provides intention actions called Create Subclass and Implement Interface respectively.

Here is an example of using the Create Subclass intention action. (The Implement Interface action is accessed in a similar way.)

1. In the editor, place the cursor within the line containing the class declaration.

2. Press `Alt+Enter` and select Create Subclass.

3. In the New ActionScript Class Dialog that opens, specify the settings for your new class and click Create.

Template-based ActionScript classes, interfaces and MXML components

ActionScript classes, interfaces and MXML components are created according to file templates . The following predefined templates are available:

- ActionScript Class. This template generates the file contents shown below (the destination package and the name of the class are specified when creating a class):

```
package myPackage {
                    public class MyClass {
                    public function MyClass() {
                    }
                    }
                    }
```

- ActionScript Class with Supers. This template generates the file contents shown below (the destination package, the class name, the superclass and/or the interfaces that the class implements are specified when creating a class):

```
package myPackage {
                    public class MyClass1 extends
                    MyClass implements IMyInterface1,
                    IMyInterface2 {
                    public function MyClass1() {
                    super();
                    }
                    }
                    }
```

– ActionScript Interface. This template generates the file contents shown below (the destination package and the interface name are specified when creating an interface):

```
package myPackage {
                    public interface IMyInterface {
                    }
                    }
```

– Flex 3 Component. This template generates the file contents shown below (the root tag is defined by the parent component specified when creating the component; when generating the following example, `mx.core.Application` was specified as the parent component):

```
<?xml version=
                    "1.0"?>
                    <mx:Application
                    xmlns:mx=
                    "http://www.adobe.com/2006/mxml">
                    </mx:Application>
```

– Flex 4 Component. This template generates the file contents shown below (the root tag is defined by the parent component specified when creating the component; when generating the following example, `spark.components.Application` was specified as the parent component):

```
<?xml version=
                    "1.0"?>
                    <s:Application
                    xmlns:fx=
                    "http://ns.adobe.com/mxml/2009"
                    xmlns:s=
                    "library://ns.adobe.com/flex/spark">
                    </s:Application>
```

If necessary, you can modify the predefined templates or create your own, custom file templates. See Creating and Editing File Templates and An example of creating a custom file template for an MXML component.

See also, Predefined file template variables for ActionScript and Flex .

Predefined file template variables for ActionScript and Flex

For ActionScript and Flex file templates, the list of predefined template variables is broader. The following predefined variables are available in addition:

– `${Superclass}` - a superclass for an ActionScript class or a parent component for an MXML component.

– `${SuperInterfaces}` - a list of the interfaces that an ActionScript class implements.

An example of creating a custom file template for an MXML component

As already mentioned, the predefined MXML 4 Component file template just generates the root tag for an MXML component. Let's assume that you want the `<fx:Declarations>` and `<fx:Script>` tags to be generated in addition.

If for some reason you want to keep the predefined template unchanged, you can create the corresponding custom file template.

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, under IDE Settings, select File Templates.
3. On the File Templates page that opens in the right-hand part of the dialog, select the Templates tab.
4. Select Flex 4 Component.
5. Now, to create a copy of this template, click .
6. Change the name of the template, for example, to Flex 4 Component with Declarations and Script.
7. In the template body (shown in the area under the Reformat according to style check box), after the line `<${Superclass} xmlns:fx="http://ns.adobe.com/mxml/2009">`

   add

```
<fx:Declarations>
                    </fx:Declarations>
                    <fx:Script><![CDATA[
                    ]]></fx:Script>
```

8. Click OK in the Settings dialog.

Now, to check the result:

1. Select your source root folder (e.g. `src`) and press `Alt+Insert`. Note that the Flex 4 Component with Declarations and Script option (which corresponds to the name of your new template) is now available in the New menu.
2. Select MXML Component. In the New MXML Component dialog that opens, check the contents of the Template list. Note that Flex 4 Component with Declarations and Script has also been added to this list.

Using the SWF metadata tag to control HTML wrapper properties

You can use the SWF metadata tag in your main application class to set the title, the background color, and the `width` and `height` properties in your HTML wrapper.

For example, if the ActionScript class contains

```
package myPackage {
                    import flash.display.Sprite;
                    [SWF(pageTitle="hello", backgroundColor="#ccddee", width="400", height="200")]
                    public class MyClass extends
                    Sprite {
                    ...
                    }
                    }
```

the corresponding properties will be set in the HTML wrapper.

In a similar way the SWF metadata tag will work for an MXML component if the corresponding file contains:

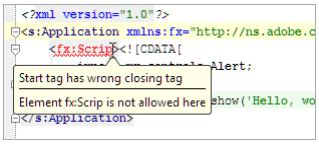
```
<?xml version=
                    "1.0"?>
                    <s:Application
                    xmlns:fx=
                    "http://ns.adobe.com/mxml/2009"
                    xmlns:s=
                    "library://ns.adobe.com/flex/spark">
                    <fx:Metadata>
                    [SWF(pageTitle="hello", backgroundColor="#ccddee", width="400", height="200")]
                    </fx:Metadata>
                    ...
                    </s:Application>
```

By default, the title is the same as the name of the embedded `.swf` file, the background color is white (`#ffffff`), and the `width` and `height` are both `100%`.

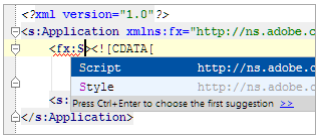
Editing ActionScript and Flex sources

When editing your ActionScript and Flex sources, you can use the following IntelliJ IDEA features:

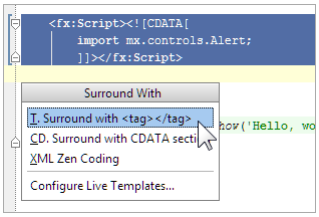
– Syntax and error highlighting. Note that the way your code is highlighted is defined by an active build configuration.



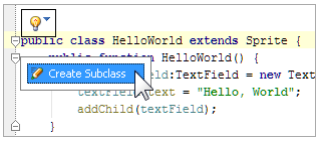
– Code completion.

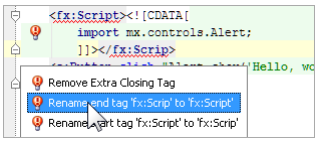


– Surrounding with tags or code constructs (`Ctrl+Alt+T` and `Ctrl+Alt+J`), and removing enclosing tags (`Ctrl+Shift+Delete`).

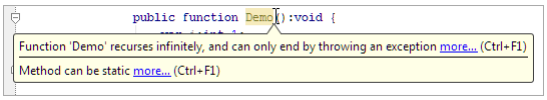


– Intention actions and quick fixes (`Alt+Enter`).

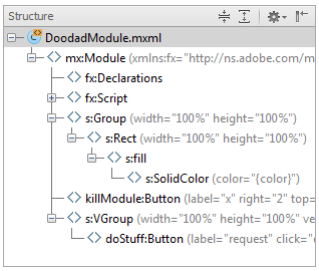




– Code inspections.



– Structure view for MXML components and ActionScript classes.



– Jumping to declarations (`Ctrl+B`).

– Code refactoring.

– Automatic code generation (`Alt+Insert`): generating getters and setters, bindable getters and setters, event handlers, etc.

Working with Build Configurations

Every Flash module in IntelliJ IDEA includes one or more build configurations. A build configuration defines how the module source files are transformed into the target output type (SWF or SWC) and then packaged.

One build configuration is created when creating a module. You can add more build configurations if and when needed.

- Build configuration types
- Main options for build configurations
- Build configuration dependencies (build path)
- Compiler options
- Active build configuration
- Managing build configurations and their settings
- Selecting an active build configuration
- Using shortcuts to open build configuration settings
- Possible changes when changing the build configuration type

Build configuration types

The build configuration type is defined by the following:

- Target platform (i.e. the environment in which the developed content is going to be used): Web (for Flash player / Web browser-targeted content), Desktop (for Adobe AIR -targeted content), or Mobile (AIR Mobile, for the content intended for Android and iOS mobile devices).
- Whether the build configuration uses the Flex framework or is pure ActionScript.
- Output type: Application (a runnable application, an SWF file), Library (an SWC file), or a Runtime-loaded module (a dynamically-loadable module , an SWF file).

Main options for build configurations

Once the build configuration type is defined, the following main options can be specified:

- For the Applications and Runtime-loaded modules: the main class .
- The output file name.
- The output folder.
- For Web Applications: the folder with the HTML wrapper template.
- For Web and Desktop Applications: the modules and runtime style sheets .

Build configuration dependencies (build path)

Generally, the build path for each build configuration may be defined by the following:

- Flex SDK. The necessary SDK SWCs are selected automatically depending on the build configuration type. Additionally, you can manage the set of the SWCs by selecting the following:
  - For the Web target platform: the Flash player version. If the SDK includes more than one player version, you can choose which of the corresponding SWCs should be used.
  - For Flex framework-based build configurations: the Flex 4 component sets. You can specify that only the Spark or MX or both component sets should be used. For Flex components you can specify their framework linkage type (Merged Into Code, RLS or External).
- Other build configurations that generate libraries and runtime-loaded modules.
- 3rd-party libraries, both SWC and raw ActionScript.

Compiler options

Each build configuration is associated with a set of compiler options.

IntelliJ IDEA provides the default sets of compiler options at the level of the IDE, project and module.

The defaults at a lower level may be inherited from the upper level or redefined. At the level of individual build configurations, similarly, the compiler options may be inherited from the module level defaults. Alternatively, build configuration-specific values may be specified.

IntelliJ IDEA provides a convenient interface for editing the defaults at various levels. There is also an ability to restore the defaults for the values that were changed.

Active build configuration

One of the build configurations is set active for the corresponding module.

The active build configuration provides the basis for source code highlighting in the editor. So when you change the active configuration, the code highlighting in the module changes as well.

Managing build configurations and their settings

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. Under Project Settings, select Modules.
3. In the area under + − | | | expand the necessary Flash module node and select any of the build configurations. Now you can:
   - View and edit the settings for the selected build configuration on the Build Configuration page shown in the right-hand part of the dialog.
   - Create a copy of the selected build configuration. To do that, click or select Copy in the context menu. Specify the settings for the copy the build configuration in the dialog that opens. Note that depending on the settings, the copy of the build configuration may have the type different from that of the original. See Possible changes when changing the build configuration type.

- Find the usages of the selected build configuration in the project. To do that, click , press `Alt+F7` or select Find usages in the context menu.
- Delete the selected build configuration. To do that, click , press `Delete` or select Delete in the context menu.
- Create a new build configuration. To do that:
  1. Click , press `Alt+Insert` or select New in the context menu.
  2. Select Flash build configuration.
  3. In the Add Build Configuration dialog that opens, specify the name and select the main options for the new build configuration, and click OK.
  4. If necessary edit the build configuration settings on the Build Configuration page .
- Change the build configuration type . To do that, click Change on the General tab (to the right of the area where the build configuration type is shown) and specify the build configuration properties in the dialog that opens. See Possible changes when changing the build configuration type.

4. Click OK in the Project Structure dialog.

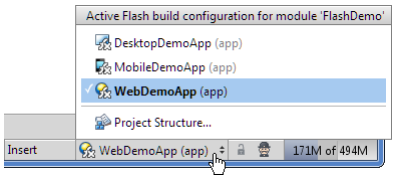
See also, Using shortcuts to open build configuration settings.

Selecting an active build configuration

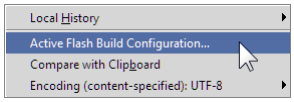
The name of the build configuration which is currently active is shown in the right-hand part of the status bar.

To make a different build configuration active, click this name.

As a result, the Active Flash build configuration menu is shown in which you can select a different build configuration which will become active.



The Active Flash build configuration menu can also be accessed from the editor when working with a `.as` or `.mxml` file. The Active Flash Build Configuration command is available for this purpose.



Using shortcuts to open build configuration settings

As already mentioned, the Active Flash build configuration menu is used to select an active build configuration (see Selecting an active build configuration).

In addition, this menu provides a shortcut for accessing build configuration settings (the Project Structure option).

When you select Project Structure in the Active Flash build configuration menu, the Project Structure dialog opens showing the settings for the active build configuration.

See also, Managing build configurations and their settings.

Possible changes when changing the build configuration type

When changing the build configuration type (by using the Change type command or when creating a copy of a build configuration), the following changes may occur if the build configuration output type has changed (e.g., from Application to Library):

- The output file name extension may change (from `.swf` to `.swc` and vice versa).
- If the output type has changed to Library, dependencies on runtime-loaded modules (if existed) are removed.
- If dependencies on the build configuration with the changed type have become inappropriate, such dependencies are removed.

When creating a copy of a build configuration, the following changes occur in addition:

- The output file name changes.
- If applicable, the package file name or names change.

Configuring dependencies for modular applications

There are two ways of configuring dependencies for modular applications. (Modular applications are ones that include dynamically loadable modules, see Modular applications overview in Flex documentation.)

One way is to list the main classes for runtime-loaded modules (RLMs) in the build configuration for the main application (a.k.a. shell). This way doesn't require creating build configurations for the modules.

The other way is to create build configurations for each of the RLMs and then, in the build configuration for the main application, specify the dependencies on these build configurations.

These two approaches along with their advantages and drawbacks are discussed below.

- Specifying the dependencies by listing the main RLM classes
- Specifying the dependencies by listing the RLM build configurations

Specifying the dependencies by listing the main RLM classes

If the source code of the main application and its runtime-loaded modules are in the same IntelliJ IDEA module, you can just list the main RLM classes in the build configuration for the main application. In addition, if you want to optimize a module against the main application (this considerably reduces the size of compiled module file), you can do that by turning the corresponding option on in the UI.

Note that this way of specifying the dependencies is available for Web and desktop applications but unavailable for mobile applications.

Advantages:

- You don't need to create build configurations for your modules (RLMs).
- To use the `load-externs` and `link-report` compiler options, you don't need to specify them manually, a check box is provided in the UI to turn module optimization on or off.

Drawbacks:

- The same set of compiler options is used for the application and the modules.
- The source code of the main application and the modules must be in the same IntelliJ IDEA module.

Here are the main steps of the procedure to be used:

1. Open the build configuration settings for the main application. For instructions, see Managing build configurations and their settings and Using shortcuts to open build configuration settings.
2. On the General tab, to the right of the Runtime-loaded modules field, click . (Alternatively, click the field and press `Shift+Enter`.)
3. In the Runtime-Loaded Modules Dialog that opens, click (`Alt+Insert`).
4. In the Choose Main Class of Runtime-Loaded Module dialog that opens, select the main class of the corresponding RLM and click OK.
5. If you want the module SWF file size to be optimized, select the Optimize check box.
6. In a similar way, add dependencies on other RLMs.
7. Click OK in the Runtime-Loaded Modules dialog.
8. Click OK in the Project Structure dialog.

Specifying the dependencies by listing the RLM build configurations

When using this way of specifying the dependencies, you should have build configurations for each of the RLMs. Besides, to optimize the file size of the RLMs, you should specify the `load-externs` and `link-report` compiler options manually (in the corresponding build configurations).

Advantages:

- The main application and the modules can be compiled using different sets of compilation options. (Every build configuration has its own set of compiler options.)
- The source code of the main application and the RLMs may be in the same IntelliJ IDEA module or in different modules (within the same project).

Drawbacks:

- You have to create a build configuration for each of the RLMs.
- To use the `load-externs` and `link-report` compiler options, you should specify them manually.

Here are the main steps of the procedure to be used:

1. Open the build configuration settings for the main application. For instructions, see Managing build configurations and their settings and Using shortcuts to open build configuration settings.
2. On the Dependencies tab, click (`Alt+Insert`), and select Build Configuration.
3. In the Add Dependency dialog that opens, select all the necessary build configurations for the RLMs and click OK.
4. If you need module optimization:
   1. On the Compiler Options tab, in the Additional compiler options field, add `-link-report=<path_to_report_file>`, for example, `-link-report=c:/temp/link-report.xml`.
   2. Add `-load-externs=<path_to_report_file>` (e.g. `-load-externs=c:/temp/link-report.xml`) to the Additional compiler options field in all the corresponding RLM build configurations.
5. Click OK in the Project Structure dialog.

Building ActionScript and Flex Applications

Your ActionScript and Flex sources are compiled according to corresponding build configurations , in particular:

- When you compile a Flash module or its part (Build | Make Module '<name>' or Build | Compile '<target_name>' (`Ctrl+Shift+F9`)), the compilation output, generally, is produced for all the build configurations associated with the module.
  To disable compilation for certain build configurations, turn on the Skip compilation option in the corresponding build configurations.
- When you compile a whole project (Build | Make Project (`Ctrl+F9`), Build | Rebuild Project), similarly, the compilation output is generated for all the build configurations for which the compilation is not explicitly disabled.
- When the sources are compiled using a run/debug configuration , the compilation output is generated only for the associated build configuration.

Compilation process

During the compilation, for each individual build configuration , the following files are processed, and the results of processing are placed into the output folder:

- Resource files (e.g., image files).
  For applications, you can explicitly specify whether you want the resource files within the module source roots to be copied to the output folder (the Copy resource files to output folder option on the Build Configuration page). You can also select individual files and folders that should not be copied (the Compiler | Excludes page of the Settings dialog).

  For libraries and RLMs, the resource files are never copied to the output folder.
- Source files (`.as` and `.mxml`).
  The source files are compiled and, depending on the build configuration output type, the corresponding `.swf` or `.swc` file is produced.
- `.css` files to be compiled into runtime style sheets (for Web and desktop applications).
  The specified `.css` files (the Runtime style sheets field on the Build Configuration page), obviously, are compiled. Their file names don't change; the extension changes to `.swf` .
- HTML wrapper template files (for Web applications).
  If so specified (the Use HTML wrapper option on the Build Configuration page), the files that constitute the HTML wrapper template are processed. (These files are stored in a separate folder called `html-template` or something similar.)

  The `index.template.html` file is renamed: the resulting `.html` wrapper file will have the same name as the application `.swf` file. The tokens contained in this file such as `${title}` , `${swf}` are replaced with the appropriate values. For example, `${swf}` is replaced with the `.swf` file name. See also, Using the SWF metadata tag to control HTML wrapper properties .

  Other files that constitute the HTML wrapper are copied to the output folder without any changes.
- An application descriptor template (for desktop and mobile applications).
  Depending on the build configuration settings, either an auto-generated descriptor is created in the output folder, or a specified template file is used. In the latter case, the text in the `<content>` element of the template is replaced with the name and extension (`.swf`) of the application file.

Running and Debugging ActionScript and Flex Applications

On this page:

- Run/debug configuration types
- Running or debugging an application from within the editor
- How IntelliJ IDEA selects or creates a class-specific run/debug configuration
- Using Flash Remote Debug configurations
- Hiding or showing [SWF] and [UnloadSWF] debugger messages

For general guidelines, see Running and Debugging .

Run/debug configuration types

The following run/debug configuration types are available for Flash modules:

- Flash App configurations let you compile and then run or debug your Flash (ActionScript and Flex) applications. You can create the necessary configurations prior to running or debugging. You can also start running or debugging an application right in the editor, when working with your source code (see Running or debugging an application from within the editor).
- FlexUnit configurations let you compile and then run or debug your FlexUnit tests. See Testing ActionScript and Flex Applications .
- Flash Remote Debug configurations let you debug applications that have already been compiled and, if necessary, packaged, and are ready to be run on a local or remote computer, or a mobile device. See Using Flash Remote Debug configurations .

To be able to debug your applications:

- The applications must be debug-ready, that is, contain the necessary debug information.
- For Web-targeted applications, you must install the debugger version of Flash player or the debugger version of the Flash player plugin for your Web browser. Normally, this software is included in Flex SDKs.
  You can also download the corresponding software separately from the Adobe Flash Player Downloads page .

Running or debugging an application from within the editor

When working with the class source code in the editor, if appropriate, you can run or start debugging your application with the current class as the main application class .
To do that, right-click somewhere in the editor area to open the context menu and select:

- Run "<class_name>" (`Ctrl+Shift+F10`) to run the application.
- Debug "<class_name>" to start debugging the application.

If a Flash App run/debug configuration appropriate for the task already exists, this configuration is selected and used. Otherwise, a new run/debug configuration is created and saved as a temporary run/debug configuration . For details, see How IntelliJ IDEA selects or creates a class-specific run/debug configuration .

In addition to Run and Debug, the following related commands may be available depending on the situation:

- Create "<class_name>". If an appropriate run/debug configuration is not found, you can use this command to create a new run/debug configuration and make it current. (The Create Run/Debug Configuration dialog will open.)
- Save "<class_name>". If the corresponding run/debug configuration is available as a temporary configuration, you can use this command to save the configuration and thus make it permanent .
- Select "<class_name>". Use this command if you want to make the corresponding temporary or permanent run/debug configuration current.

All the functions described above may alternatively be accessed as the context menu commands in the Project or Favorites tool window.

How IntelliJ IDEA selects or creates a class-specific run/debug configuration

As already mentioned, when you run or debug your application from within the editor , IntelliJ IDEA first tries to find an existing run/debug configuration with the class you are currently working with as the main class.

If no such configuration is found, a new run/debug configuration is created.

If more than one configuration with the suitable main class is found, IntelliJ IDEA prioritizes the configurations according to the following conditions and select the one with the highest priority:

1. The run/debug configuration is based on the active build configuration, the build configuration output type is Application, the main class is not overridden in the run/debug configuration.
2. The run/debug configuration is based on a build configuration with the output type Application, the main class is not overridden in the run/debug configuration.
3. The run/debug configuration is based on the active build configuration, the main class is overridden in the run/debug configuration.
4. Any run/debug configuration with the suitable (overridden) main class.

When creating a new run/debug configuration, IntelliJ IDEA tries to find a build configuration with a suitable main class. If found, the new run/debug configuration will be based on such a build configuration. Otherwise, the active build configuration will be used, and the main class will be overridden in the new run/debug configuration.

Using Flash Remote Debug configurations

1. Select and start the necessary Flash Remote Debug configuration (or `Shift+F9`).
   As a result, the Debug tool window opens; the debugger is waiting for the application to connect.
2. Now, to connect to the debugger, do one of the following:
   - Start the Flash or AIR application on your local computer; the application will connect to the debugger automatically.
   - If the application is already running in a Web browser or a Flash player on the local or remote computer, right-click the corresponding page in the browser or the application in the player, and select Debugger from the context menu.
   - Start the application on the mobile device. If the device is capable of communicating with your computer, the application will connect to the debugger automatically.

Hiding or showing [SWF] and [UnloadSWF] debugger messages

Your Flash Player Debugger output may contain many `[SWF]` and `[UnloadSWF]` messages (the Console tab of the Debug tool window). There are situations when you don't want to see them.

You can hide or show the `[SWF]` and `[UnloadSWF]` messages by using on the toolbar.

Note that changing the state of this toggle doesn't change the current console content. That is, if you turn the filter on, the messages already present in the output won't be hidden. Only new messages won't be shown.

Packaging AIR Applications

When ready, you can package your AIR (Desktop and Mobile) applications for all the appropriate build configurations existing in the project.

To package AIR applications

1. Choose Build | Package AIR Application
2. In the Package AIR Application dialog that opens, select the necessary build configurations, specify packaging options and click Package.

Importing Adobe Flash Builder Projects

You can import Adobe Flash Builder projects and then continue working with them in IntelliJ IDEA.

Each individual Flash Builder project imported into IntelliJ IDEA is represented by a Flash module .

By importing Flash Builder projects, you can create a new IntelliJ IDEA project . You can as well import Flash Builder projects into an existing IntelliJ IDEA project.

For creating a new project, two options are available. You can use the Import Project or the Open command. Importing is more precise but longer, opening is quicker but less precise.

– Supported Flash Builder formats
– Creating an IntelliJ IDEA project by using the Import Project command
– Creating an IntelliJ IDEA project by using the Open command
– Importing Flash Builder projects into an existing project

Supported Flash Builder formats

When importing Flash Builder projects into IntelliJ IDEA, you can specify the following as a source of import:

– A Flash Builder workspace or project directory
– A `.project` file
– A `.fxp` or `.fxpl` file
– A `.zip` file that contains an ActionScript project or projects.

Creating an IntelliJ IDEA project by using the Import Project command

1. If no project is currently open in IntelliJ IDEA, click Import Project on the Welcome screen . Otherwise, select File | New | Project from Existing Sources.
2. In the dialog that opens , select the directory or file which you want to use as a source of import . Click OK.
   As a result, the Import Project wizard opens.
3. On the first page of the wizard , select Import project from external model, then select Flash Builder and click Next. (This page is not shown if IntelliJ IDEA has guessed that you are importing a Flash Builder project.)
4. On the next page of the wizard:
   1. Specify the location of the file or directory to be imported . Use … to select the necessary file or folder in the corresponding dialog .
   2. If necessary, change the name and location of the IntelliJ IDEA project that is going to be created.
   3. Optionally, select the Create subfolder option (if present).
   4. If necessary, change the suggested project format .
   5. Click Next or Finish.
5. If you are importing a workspace, select the projects that you want to import on the next page of the wizard and click Finish.
6. If undefined path variables are found in the project or projects that you are importing, the Configure Path Variables dialog is shown. Specify the values of the undefined variables or include them in the list of ignored variables.
7. In the Flash Builder Project Import dialog, specify the Flex SDK to be associated with the imported project or projects in IntelliJ IDEA.

Creating an IntelliJ IDEA project by using the Open command

1. If no project is currently open in IntelliJ IDEA, click Open on the Welcome screen . Otherwise, select File | Open.
2. In the dialog that opens , select a file or directory that you want to use as a source of import , and click OK.
3. If suggested, specify the location of the IntelliJ IDEA project that is going to be created (shown under Extract project to in the corresponding dialog).
4. If undefined path variables are found in the project or projects that you are importing, the Configure Path Variables dialog is shown. Specify the values of the undefined variables or include them in the list of ignored variables.
5. In the Flash Builder Project Import dialog, specify the Flex SDK to be associated with the imported project or projects in IntelliJ IDEA.

Importing Flash Builder projects into an existing project

1. Open the project into which you want to import Flash Builder projects, and select File | New | Module from Existing Sources.
2. In the dialog that opens , select the directory or file which you want to use as a source of import . Click OK.
   As a result, the Import Module wizard opens. (There is also another way of accessing this wizard, see Opening the Import Module wizard.)
3. On the first page of the wizard , select Import module from external model, then select Flash Builder and click Next. (This page is not shown if IntelliJ IDEA has guessed that you are importing a Flash Builder project.)
4. On the next page of the wizard:
   1. Specify the location of the file or directory to be imported . Use … to select the necessary file or folder in the corresponding dialog .
   2. If necessary, change the location of the module that is going to be created. (Shown under Extract project to; the corresponding option may be missing.)
   3. Click Next or Finish.
5. If you are importing a workspace, select the projects that you want to import on the next page of the wizard and click Finish.
6. If undefined path variables are found in the project or projects that you are importing, the Configure Path Variables dialog is shown. Specify the values of the undefined variables or include them in the list of ignored variables.

7. In the Flash Builder Project Import dialog, specify the Flex SDK to be associated with the imported project or projects in IntelliJ IDEA.

Testing ActionScript and Flex Applications

IntelliJ IDEA supports the versions 0.9 and 4 of FlexUnit , a unit testing framework for Flex and ActionScript applications and libraries.

For running FlexUnit tests, there is a dedicated run/debug configuration type (FlexUnit).

The FlexUnit-specific tasks are briefly outlined below. For general instructions, see Testing .

– Preparing for writing FlexUnit tests

– Running or debugging FlexUnit tests

Preparing for writing FlexUnit tests

To prepare for writing FlexUnit tests, you should add the core FlexUnit library to dependencies of the corresponding build configuration , and make this build configuration active .

1. Download and decompress the archive containing FlexUnit libraries. (The corresponding archives (`.zip`) are available, for example, on the FlexUnit Downloads page . The archive name, normally, starts with `flexunit` .)
   The only file that you'll need is the core FlexUnit library (`.swc`). Depending on the FlexUnit version, this file may be called `FlexUnit.swc` , `flexunit-4.0.0.swc` , `flexunit-4.1.0-8-as3_4.1.0.16076.swc` , `flexunit-core-flex-4.1.0-beta1.64-sdk4.0.0.14159.swc` or something similar.

2. Though this isn't really necessary, it might be useful to define the core FlexUnit library as a global or project library . This will let you use this library in many projects or modules. See Configuring Project and Global Libraries .

3. Add the core FlexUnit library to dependencies of one or more of the build configurations. See Managing build configurations and their settings and Build Configuration Page for a Flash Module .

4. Make one of these build configurations active. This will ensure that the code in your tests is highlighted properly. See Selecting an active build configuration.

Now you are ready to start developing your test sources.

Running or debugging FlexUnit tests

To run or debug your FlexUnit tests, you can create the necessary FlexUnit run/debug configurations and then use those. Note that the corresponding run/debug configurations should be based on the build configurations that have the core FlexUnit library among their dependencies. See Creating and Editing Run/Debug Configurations and Run/Debug Configuration: FlexUnit .

The other alternative is to run or debug you test package, class or method from within the editor, or the Project or Favorites tool window. (The Run and Debug commands, if appropriate, are available in the context menus.) In such cases you don't even need to create the run/debug configuration prior to running the tests.

ActionScript-Specific Refactorings

In this section:

- Change Method Signature in ActionScript
- Extract Parameter in ActionScript

For the complete list of the refactorings available for ActionScript, see ActionScript and Flex .

Change Method Signature in ActionScript

In ActionScript, you can use the Change Method Signature refactoring to:

- Change the method name and return type.
- Add new parameters and remove the existing ones. Note that you can also add a parameter using a dedicated Extract Parameter refactoring.
- Reorder parameters.
- Change parameter names and types.
- Propagate new parameters through the method call hierarchy.

On this page:

- Example
- Initializer, default value, and propagation of new parameters
- More refactoring examples
- Changing a method signature

Example

Before After

```
// The function paint() is declared in
                    // the IShape interface.



                    public interface IShape {
                    function paint(g:
                    Graphics): void;
                    }

                    // This function is then called within the
                    // paint() function of the Canvas class.



                    public class Canvas {
                    private var shapes: Vector.<IShape>;

                    public function paint(g: Graphics): void
                    {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g);
                    }
                    }
                    }

                    // Now we are going to show an example of the
                    // Change Signature refactoring for the function
                    // paint() of the IShape interface.
```

```
// In this refactoring example we have changed the name of the existing
                    parameter
                    // and introduced two new parameters. Note that the first of the new parameters
                    // a required parameter while the second is optional because the default value
                    // for it is specified in the function definition.

                    public interface IShape {
                    function paint(graphics:Graphics, wireframe:Boolean, offset:Point
                    = null):void;
                    }

                    // When performing this refactoring, the new parameters were propagated to
                    // the paint() function of the Canvas class. As a result, the signature of
                    // Canvas.paint() has changed. Also note how IShape.paint() within
                    // Canvas.paint() is called now.

                    public class Canvas {
                    private var shapes: Vector.<IShape>;

                    public function paint(g:Graphics, wireframe:Boolean): void {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g, wireframe);
                    }
                    }
                    }

                    // Other results for this refactoring are possible.
                    // For more information, see the discussion that follows.
```

Initializer, default value, and propagation of new parameters

For each new parameter added to a function, you can specify:

- A value (or an expression) to be used for initializing the parameter (the Initializer field in IntelliJ IDEA).
- A default value (or an expression) (the Default value field).

You can also propagate the parameters you have introduced to the functions that call the function whose signature you are changing.

The refactoring result depends on whether or not you specify those values and use the propagation.

Propagation. New parameters can be propagated to any function that call the function whose signature you are changing. In such a case, generally, the signatures of the calling functions change accordingly. These changes, however, also depend on the combination of the initializer and the default value set for the new parameters.

Initializer. The value specified in the Initializer field is added to the function definition as the default parameter value. This makes the corresponding parameter an optional parameter. (See the discussion of required and optional parameters in Flex/ActionScript documentation .)

If the default value for the new parameter is not specified (in the Default value field), irrespective of whether or not the propagation is used, the function calls and the signatures of the calling functions don't change.

If both, the initializer and the default value are specified, the refactoring result depends on whether or not the propagation is used:

- If the propagation is not used, the initializer value don't affect the function calls and the signatures of the calling functions.
- If the propagation is used, the initializer value is added to the definition of the calling function as the default value for the corresponding parameter (in the same way as in the function whose signature you are changing).

Default value. Generally, this is the value to be added to the function calls.

If the new parameter is not propagated to a calling function, the function calls within such a function will also use this value.

If the propagation is used, this value won't matter for the function calls within the calling functions.

More refactoring examples

To see how different refactoring settings discussed above affect the refactoring result, let us consider the following examples.

All the examples are a simplified version of the refactoring shown earlier . In all cases, a new parameter `wireframe` of the type `Boolean` is added to the function `paint()` defined in the `IShape` interface.

In different examples, different combinations of the initializer and the default value are used, and the new parameter is either propagated to `Canvas.paint()` (which calls `IShape.paint()`) or not.

Initializer	Default value	Propagation used	Result
	`false`	Yes	

```
public interface IShape
                    {
                    function paint(g:Graphics,
                    wireframe:Boolean):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics,
                    wireframe:Boolean): void
                    {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g, wireframe);
                    }
                    }
```

	`false`	No	

```
public interface IShape
                    {
                    function paint(g:Graphics,
                    wireframe:Boolean):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics): void
                    {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g, false);
                    }
                    }
```

`true`		Yes	

```
public interface IShape
                    {
                    function paint(g:Graphics,
                    wireframe:Boolean = true):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics): void
                    {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g);
                    }
                    }
```

`true`		No	

```
public interface IShape
                    {
                    function paint(g:Graphics,
                    wireframe:Boolean = true):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics): void
                    {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g);
                    }
                    }
```

`true`	`false`	Yes	

```
public interface IShape
                    {
                    function paint(g:Graphics,
                    wireframe:Boolean = true):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics,
                    wireframe:Boolean = true): void {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g, wireframe);
                    }
                    }
```

`true`	`false`	No

```
public interface IShape
                    {
                    function paint(g:Graphics,
                    wireframe:Boolean = true):void;
                    }

                    // The function paint() in the Canvas class:

                    public function paint(g:Graphics): void
                    {
                    for each (var shape: IShape in shapes)
                    {
                    shape.paint(g, false);
                    }
                    }
```

Changing a method signature

1. In the editor, place the cursor within the name of the method whose signature you want to change.

2. Do one of the following:
   - Press `Ctrl+F6`.
   - Choose Refactor | Change Signature in the main menu.
   - Select Refactor | Change Signature from the context menu.

3. In the Change Signature dialog , make the necessary changes to the method signature and specify which other, related changes are required.
   You can:
   - Change the method return type by editing the contents of the Return type field.

     Note Code completion is available in this field and also in certain fields of the table that contains the function parameters.
   - Change the method name. To do that, edit the text in the Name field.
   - Manage the method parameters using the table of parameters and the buttons to the right of it:
     - To add a new parameter, click + (`Alt+Insert`) and specify the properties of the new parameter in the corresponding fields.
       When adding parameters, you may want to propagate these parameters to the methods that call the current method.
     - To remove a parameter, click any of the cells in the corresponding row and click — (`Alt+Delete`).
     - To reorder the parameters, use ↑ (`Alt+Up`) and ↓ (`Alt+Down`). For example, if you wanted to make a certain parameter the first in the list, you would click any of the cells in the row corresponding to that parameter, and then click ↑ the required number of times.
     - To change the name or type for a parameter, make the necessary edits in the corresponding table cells.
   - Propagate new method parameters (if any) along the hierarchy of the methods that call the current method.
     (There may be the methods that call the method whose signature you are changing. These methods, in their turn, may be called by other methods, and so on. You can propagate the changes you are making to the parameters of the current method through the hierarchy of the calling methods and also specify which calling methods should be affected and which shouldn't.)

     To propagate the new parameters:
     1. Click (`Alt+G`).
     2. In the left-hand pane of the Select Methods to Propagate New Parameters dialog, expand the necessary nodes and select the check boxes next to the methods you want the new parameters to be propagated to.
        To help you select the necessary methods, the code for the calling method and the method being called is shown in the right-hand part of the dialog (in the Caller Method and Callee Method panes respectively).

        As you switch between the methods in the left-hand pane, the code in the right-hand part changes accordingly.
     3. Click OK.

4. To perform the refactoring right away, click Refactor.
   To see the expected changes and make the necessary adjustments prior to actually performing the refactoring, click Preview.

Extract Parameter in ActionScript

This section discusses the Extract Parameter refactoring in ActionScript.

– Example

– Introducing a parameter in ActionScript

Example

Before After

```
// Two ways of introducing a parameter for the function
                    // formatPrice() will be shown.

                    public function foo():void {
                    formatPrice(0);
                    }

                    // The new parameter will be optional in the first
                    // of the examples and required in the second example.

                    public function formatPrice(value:int):String
                    {
                    trace("currency: " +
                    "$");
                    return
                    "$" + value;
                    }
```

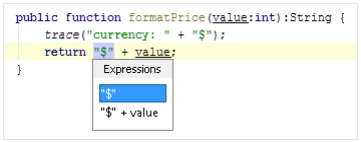
```
// The function formatPrice() may be called as before because
                    // the new parameter is introduced as an optional parameter.

                    public function foo():void {
                    formatPrice(0);
                    }

                    // The default value for the new parameter is specified
                    // in the function definition.

                    public function formatPrice(value:int, s:String = "$"):String {
                    trace("currency: " +
                    s);
                    return
                    s + value;
                    }


                    // The new parameter in this example is a required parameter.
                    // So two values must be passed to formatPrice() now.

                    public function foo():void {
                    formatPrice(0, "$");
                    }

                    // The new parameter is a required parameter because the default
                    // value for it is not specified in the function definition.

                    public function formatPrice(value:int, s:String):String {
                    trace("currency: " +
                    s);
                    return
                    s + value;
                    }
```

Introducing a parameter in ActionScript

1. In the editor, place the cursor within the expression to be replaced by a parameter.

2. Do one of the following:
   – Press `Ctrl+Alt+P`.
   – Choose Refactor | Extract | Parameter in the main menu.
   – Choose Refactor | Extract | Parameter from the context menu.

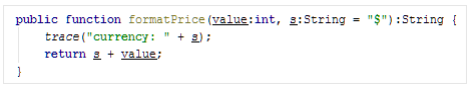
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



4. In the Extract Parameter dialog :
   1. Usually, IntelliJ IDEA sets a proper parameter type itself. If necessary, you can select another appropriate type from the Type list.
   2. Specify the parameter name in the Name field.
   3. The Value field, initially, contains the expression that you have selected. Normally, you don't need to change this.
      If the new parameter is going to be an optional parameter, the specified value will be used as the default parameter value in the function definition.

      If the new parameter is introduced as a required parameter, the specified value will be added to the function calls.

      For information about required and optional parameters, see the discussion of function parameters in Flex/ActionScript documentation .
   4. If you want the new parameter to be an optional parameter, select the Optional parameter check box.
   5. If more than one occurrence of the expression is found within the function body, you can choose to replace only the selected occurrence or all the found occurrences with the references to the new parameter. Use the Replace all occurrences check box to specify your intention.
   6. Click OK.



Android

In this part:

- Android Support Overview
- Getting Started with Android Development
- Creating Android Application Components
- Managing Resources
- Designing Layout of Android Application
- Running and Debugging Android Applications
- Testing Android Applications
- Sharing Android Source Code and Resources Using Library Projects
- Renaming an Application Package (Application ID)
- Generating Signed and Unsigned Android Application Packages
- Accessing Android SQLite Databases from IntelliJ IDEA
- Android Tutorials

Android Support Overview

To support Android development, IntelliJ IDEA provides a dedicated Android Support plugin. This plugin is bundled with the IDE and is enabled by default.

Android support in IntelliJ IDEA includes:

- Gradle-based build system for Android applications.
- Smart XML editor that facilitates working with layout, manifest, resources and other XML files.
- A graphical editor for layout files: build the design of your application without having to edit the XML files manually.
- Enhanced navigation between related files: navigate between layout files and the associated activities, or a service and the manifest file, etc. via gutter icons.
- Automatic generation of basic layout files .
- Android-specific code inspections , including integrated Android Lint inspections.
- Possibility to preview your application UI on different devices at the same time.
- Access to SQLite Databases right from the IDE.
- Built-in 9-patch editor .
- Detection of mismatched resource types in Android APIs and your own libraries and APIs with the help of resource type annotations .
- Possibility to automatically rearrange XML attributes according to predefined rules.
- Dedicated Android run/debug configurations .
- Dedicated Android artifacts .

Getting Started with Android Development

In this topic:

- Before you start
- Choosing the module type you need
- Creating Android Project
- Adding an Android module to a project
- Attaching an Android facet to an existing Java module
- Configuring the code style of Android-specific XML definition files

Before you start

- Download and extract the Android SDK . We strongly recommend that the path to the Android SDK home directory does not contain spaces.
  Note that the Android SDK is not a substitute for a Java SDK (JDK). You need to download and configure a Java SDK for your project anyway.
- Add SDK packages. For detailed instructions and download links, refer to Adding SDK Packages .
- Configure the Android SDK in IntelliJ IDEA, see Configuring Global, Project and Module SDKs .
- Make sure that the bundled Android Support plugin is activated. This plugin provides Android support at the IntelliJ IDEA level and is enabled by default. If not, enable it as described in the section Enabling and Disabling Plugins .
- Depending on your task, choose the module type you need for Android development.
- Create a project with an Android module from scratch, add an Android module to a project, or attach an Android facet to an existing Java module.
  Depending on the chosen module type, IntelliJ IDEA automatically sets up the correct module structure with the `res` and `gen` folders, downloads the necessary libraries, and generates various Android-specific descriptors.

Note Starting with the IntelliJ IDEA version 2016.1, non-Gradle Android projects are not supported.

Choosing the module type you need

IntelliJ IDEA lets you choose Application modules (form factors) when you create your project, add different Android modules to an existing project and add Android facets to your modules.

Creating Android Project

1. Open Project Wizard , in the left-hand pane select Android .
2. In the right-hand pane configure your new project and click Next.
3. On the next page of the wizard, select the application module type and its minimum SDK. Click Next.
4. On the next page of the wizard, select the Android activity and click Next.
5. On the next page of the wizard, specify the Activity settings and click Finish.

Adding an Android module to a project

1. Choose File | New | Module from the main menu or New | Module from the context menu of the Project tool window.
   The New Module wizard opens.
2. In the left-hand pane, select Android. In the right-hand pane, select a module you want to add and click Next.
3. On the next page of the wizard, specify the application or library name, module name, package name and a minimum SDK that is required for this type of application. Click Next.
4. On the next page of the wizard, select an activity for your module and click Next.
5. On the next page of the wizard, specify the settings to customize the selected activity and click Finish.

Attaching an Android facet to an existing Java module

You need to perform this procedure if you want to attach an Android facet to an existing Java module. Android modules have this facet applied by default.

1. Open the Project Structure dialog box.
2. Under Project Settings, select Modules.
3. Select the module you want to add an Android facet to, click +, and choose Android.
4. On the Facet 'Android' page that opens, specify the location of the key application components in the Structure tab: the AndroidManifest.xml file, the application resources , the application assets , and the Android native libraries. .
   If necessary, you can edit the default paths. To return to the default Android facet settings, click Reset paths to defaults.
5. To make the module source code and resources available from other projects , select the Library module check box on top of this page.

Configuring the code style of Android-specific XML definition files

Android development involves working with dedicated XML files, such as layout and resource definition files, manifest files, etc. You can have IntelliJ IDEA apply the standard XML code style to such files, or configure custom code style settings for them.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click XML under Code Style.
2. On the Code Style:XML page that opens, switch to the Android tab.
3. Do one of the following:
   - To define a custom code style for Android-specific XML files, select the Use custom formatting settings for Android XML files check box and configure the settings to be applied to various types of Android XML files using the controls of the tab as described in Code Style:XML - Android .
   - To have IntelliJ IDEA format Android-specific XML files according to the standard XML code style settings defined in the other tabs of the page, clear the Use custom formatting settings for Android XML files check box.

Creating Android Application Components

An Android application may contain the following components:

- AIDL : an Android Interface Definition Language (AIDL) interface used for interprocess communication.
- Activity : implements a window where you place your UI to interact with the user.
- Android Auto : lets you extend your application for use in vehicles. You can add either Media Service or Messaging Service activity.
- Folder: creates a source root based on the activity you have selected for this component.
- Fragment : represents a behavior or a part of user interface in an activity.
- Google: lets you create an activity for Google maps and AdMob Ads activities.
- Application : an Android package, i.e. an `.apk` archive that contains the contents of an Android app and the installer.
- Service : represents an application's desire either to perform an operation without interacting with the user, or to supply functionality for other applications.
- Other: lets you add the following components to your application:
  - Android Manifest File
  - Broadcast Receiver
  - Content Provider
  - Daydream
- UI Component : lets you add custom views to you application.
- Wear : lets you extend your application for use in Android wear.
- Widget : lets you add different types of widgets for your application.
- XML : lets you add different types .xml files for Android layouts and values.

All Android application components are created in the same way. When you create a new component, a class that implements this component is generated and the component is automatically declared in the AndroidManifest.xml file.

On this page:

- Creating an Android component
- Navigating between an activity or a fragment and its related layout definition file
- Navigating from a component to its declaration in the AndroidManifest.xml file

Creating an Android component

1. In the Project view, right-click the destination package where the application classes are stored and from the context menu, select New | 'component name' and the specific service, file or an activity .
2. In the dialog box that opens , specify the necessary information and click Finish.

Note that when you create an activity, a service or a broadcast receiver, they are automatically registered in the `androidmanifest.xml` file.

Navigating between an activity or a fragment and its related layout definition file

You can jump from the source code of an activity or a fragment to the layout definition file which represents its content view and vice versa, from the layout definition to the source code.

- To jump from a component to its related layout definition, open the source code of the component, and do one of the following:
  - Click the icon in the gutter area and choose the layout definition file in the Go To Related Symbol pop-up list.
  - On the main menu, choose Navigate | Related Symbol.
- To jump from a layout definition to the source code of the corresponding component, open the layout definition file in the text mode and do one of the following:
  - Click the icon in the gutter area.
  - On the main menu, choose Navigate | Related Symbol.

Navigating from a component to its declaration in the AndroidManifest.xml file

As soon as a component is created, it is immediately declared in the `AndroidManifest.xml` file. For components of the activity, fragment, service, and broadcast receiver type, you can jump to this declaration right from the component source code.

1. Open the source code of the component, and click the icon in the gutter area.
2. If the component also has a related layout definition file, choose AndroidManifest.xml in the Go To Related Symbol pop-up list.

Managing Resources

IntelliJ IDEA provides a number of facilities for managing resources used in Android applications. In the project view, resource definition files are shown grouped into folders under the `res` directory.

The following coding assistance is provided:

- Basic code completion
- Code completion for various resource types, including alternate resources
- Syntax and error highlighting
- Find usages
- Navigation from resource usages to resource definition
- Quick fixes including resource creation upon reference failure
- Code completion for referencing Android SDK resources
- Extract style refactoring (see Refactoring Android XML Layout Files for details)
- Extract layout refactoring (see Refactoring Android XML Layout Files for details)
- Extract string resource intention (see Refactoring Android XML Layout Files for details)
- Translation editor
- For drawable resources, a preview icon is displayed in the `.java` and `AndroidManifest.xml` files in the left editor gutter next to this resource reference in the code.

Creating Resources

Besides the standard way to create files and folders , you can create resource definitions right from the `res` node without having to place them in the corresponding folder. IntelliJ IDEA detects the type and the qualifier of the new resource and appoints the folder where the resource definition file will be saved. You can create separate resource definition files and entire folders.

In this topic:

- Creating a resource folder
- Creating a resource definition file in the relevant resource folder

Creating a resource folder

IntelliJ IDEA composes the name for the new resource folder from the resource type and the qualifiers you choose, and creates a folder with this name under the `res` folder. The name is built in compliance with the Android naming conventions.

1. In the Project tool window, right-click the `res` node where all Android resource definitions are stored. Select New | Android resource directory from the context menu. The New Resource Directory dialog box opens.

2. From the Resource Type drop-down list, select the type of resources to be stored in the new folder. The selected type is displayed in the Directory name field.

3. Do one of the following:
   - If your application does not need to be compatible with various Android devices and, therefore, no multiple screens support is required, just click OK.
   - To provide alternative resources , specify the resource qualifiers that define a specific device configuration.
     - From the Available Qualifiers list, select the required qualifier, and click the » button. The selected item is added to the Chosen qualifiers list. Depending on a particular qualifier, either specify its value manually, or choose the required value from the drop-down list next to Chosen qualifiers.
     - To remove a qualifier, select it in the Chosen qualifiers list and click the « button.

     IntelliJ IDEA appends all selected qualifiers to the resource type with the dash character as a separator. The resulting name is shown in the Directory name field.

IntelliJ IDEA generates the new folder with the compound name under the `res` node.

Creating a resource definition file in the relevant resource folder

1. In the Project tool window, right-click the `res` node where all Android resource definitions are stored. Select New | Android resource file from the context menu. The New Resource File dialog box opens.

2. Specify the name for the new resource definition file in the File name text box. Follow the Android naming guidelines

3. From the Resource Type drop-down list, select the resource type . You can scroll through the list of resource types right from the `File name` text box by using the Up and Down keyboard keys.

4. To enable coding assistance, in the Root element field, specify the `http://schemas.android.com/apk/res/android` schema as the value of the `xmlns:android` .

5. Do one of the following:
   - If your application does not need to be compatible with various Android devices and, therefore, no multiple screens support is required, just click OK.
   - To provide alternative resources , specify the resource qualifiers that define a specific device configuration.
     - From the Available Qualifiers list, select the required qualifier, and click the » button. The selected item is added to the Chosen qualifiers list. Depending on a particular qualifier, either specify its value manually, or choose the required value from the drop-down list next to Chosen qualifiers.
     - To remove a qualifier, select it in the Chosen qualifiers list and click the « button.

     IntelliJ IDEA appends all selected qualifiers to the resource type with the dash character as a separator. The resulting name is shown in the Directory name field.

6. Click OK, when ready. IntelliJ IDEA composes the name of the resource folder where the new file is going to be saved. The name is built from the specified resource type and the qualifier in compliance with the Android naming conventions. If such folder exists already, IntelliJ IDEA saves the new file in it. If no such folder is found, IntelliJ IDEA creates it under the `res` node and saves the new resource definition file there.

Creating Image Assets

IntelliJ IDEA allows you to easily create icons for your Android applications with the help of the Asset Studio wizard. The wizard creates multiple icons for different screen resolutions and lets you see a live preview in the process of creating. You can create icons using your own images, clipart images, or text, configure the background shape, the colors, the fonts, etc.

To create an image asset

1. In the Project tool window, right-click the `res` where all Android resource definitions are stored. Select New | Image Asset from the context menu. The Asset Studio wizard opens.
2. On the first page of the wizard , select the asset type (launcher icons, action bar and tab icons, or notification icons) and the source for your icon: image, clipart or text.
3. Adjust your icon's shape, colors, aspect ratio, etc. depending on the selected asset type and source (for a detailed explanation of the available options, refer to Asset Studio. Page 1 .
4. On the second page of the wizard , preview output formats of your image asset and click Finish to complete the wizard.

Designing Layout of Android Application

A layout defines the user interface of an activity or an app widget (fragment). Layouts are declared in XML resource definition files. See Creating Resources for instructions on how to create resource folders and resource definition files.

In this topic:

- Layout Editing Modes
- Toggling between the Design and the Text Modes
- Navigating between an activity or a fragment and its related layout definition file

Layout Editing Modes

IntelliJ IDEA suggests two main ways to design the user interface of your Android application:

- Edit the layout definition files manually, possibly using the Android-specific refactoring provided by IntelliJ IDEA, and preview the changes that are immediately reflected in the dedicated Preview tool window, where you can adjust the layout to various platforms and devices.
- Compose the layout in the dedicated Designer tool window: drag and drop layout elements from the Palette pane and specify their properties in the Properties pane. In this mode, all changes are also reflected immediately in the preview that is displayed right in the editor.
  At the IntelliJ IDEA level, this functionality is provided through the bundled Android Support plugin, which is enabled by default. If not, enable it in the Plugins page of the Settings/Preferences dialog box.

Toggling between the Design and the Text Modes

You can toggle between these modes by switching between the Design and Text tabs in the editor where the layout definition file is opened.

Note, that the set of available panes and tool windows depends on the current layout editing mode: the Designer tool window with its panes is available in the Design mode, while the Preview tool window is available in the Text mode.

In the Design mode, you can switch to the manual mode by choosing Go to Declaration on the context menu of the Design pane, or by clicking the icon in the Preview window Toolbar .

Navigating between an activity or a fragment and its related layout definition file

You can jump from the source code of an activity or a fragment to the layout definition file which represents its content view and vice versa, from the layout definition to the source code.

- To jump from a component to its related layout definition, open the source code of the component, and do one of the following:
  - Click the icon in the gutter area and choose the layout definition file in the Go To Related Symbol pop-up list.
  - On the main menu, choose Navigate | Related Symbol.
- To jump from a layout definition to the source code of the corresponding component, open the layout definition file in the text mode and do one of the following:
  - Click the icon in the gutter area.
  - On the main menu, choose Navigate | Related Symbol.

Adding and Editing Layout Components Using Android UI Designer

A layout defines the user interface of an activity or an app widget (fragment). Layouts are declared in XML resource definition files. See Creating Resources for instructions on how to create resource folders and resource definition files.

With IntelliJ IDEA, you can build the design of your application without editing the layout definition files manually, and check how the application design is rendered in various target environments without running the application on any physical or virtual devices.

> **Note** Alternatively, edit the layout definition files manually, possibly using the Android-specific refactoring provided by IntelliJ IDEA, and preview the changes that are immediately reflected in the dedicated Preview tool window, where you can adjust the layout to various platforms and devices. To switch to the manual mode, click the Text tab or choose Go To Declaration from the context menu in the Design pane, or click the icon in the Preview window Toolbar.

Designing your layout in the visual mode is performed in the Design pane of the Android UI Designer tool window. The pane is located in the central part of the UI Designer (assuming the default tool window layout). When you open a layout definition file for editing, the pane appears in the editor tab by default. If you are editing the layout definition file manually and then switch to the visual mode by clicking the Design tab, the pane opens in the tab where the edited layout definition file is opened,

The pane shows a rectangular canvas that is synchronized with the current layout definition file and with the Component Tree view, so any changes to the canvas are reflected there accordingly. If IntelliJ IDEA detects any discrepancy in the code, a warning is displayed.

Note that intention actions and quick fixes are available in the design mode in the same way as when you edit the layout definition files manually. The intetion action icon or the quick-fix icon is displayed on the canvas, in the Component Tree or in the Properties pane.

To build the design of an Android application, perform the following basic operations:

- Add predefined components from the Palette
- Add user-defined components and components from Android SDK
- Arrange the components
- Specify component properties
- Convert components into other types preserving the common properties
- Preview the layout

To add a predefined component to the canvas

Do one of the following:

- Select the required element in the Palette pane and drag and drop it to the canvas in the Design pane.
- Click the required element in the Palette pane and then click an area on the canvas.
- Click the required element in the Palette pane and then click the Component Tree where you want this element to be positioned.

Every component added in either way is also added to the Component Tree and is declared in the layout definition file.

To add a component defined either in your project or in the Android SDK

1. Expand the Custom area in the Palette pane.
2. Do one of the following:
   - To embed a layout, click the Include icon. In the Resources dialog box that opens, click the Project tab to search among your layout definitions or click the System tab to search in the SDK.
   - To add a combination of components, click Fragment and choose the fragment to embed in the Resources dialog box that opens.
   - To add a user-defined view, click CustomView and choose the view to insert in the dialog box that opens.

To place a component in the right position

The canvas is synchronized with the Component Tree, so you can arrange components by moving them in either of these in either pane.

Select the required component on the canvas or in the Component Tree and drag it to the right position or copy and paste it using the context menu.

To specify component properties

You can set the values for the mandatory component properties right in the canvas, or switch to the Properties pane for a more in-depth configuration.

- To assign the values for the essential properties in the canvas, double-click the component in question and specify the values in the pop-up dialog box that appears.
- For an in-depth configuration of the component properties, select the component in the canvas or in the Component Tree, switch to the Properties pane, and specify the values for the properties of your choice.
  Click the right column in the Properties pane to start editing a property. You can click the Browse button the appears on the right to select or project or a system resource.
  - By default, the pane shows only a standard set of properties, and the most frequently used ones are displayed in bold. To have the pane display all properties that are defined for the selected component according to the specification, click the Show expert properties button on the toolbar.
  - To view a brief documentation for the selected property from Android reference, click the Show documentation button on the toolbar or press `Ctrl+Q`.
  - Properties with updated values are highlighted in blue. To discard the changes you have made and return to the default value, select the property and click the Restore default value button on the toolbar.

To convert a component into another type

In some cases, you may need to transform an already fully configured component into a component of another type. With IntelliJ IDEA, you can do it without

losing the specified properties: all properties that are common for both types will be preserved in the new component. This operation is referred to as morphing.

1. Select the component that you want to convert in the canvas or in the in the Component Tree and choose Morphing on the context menu of the selection.
2. IntelliJ IDEA displays the list of compatible component types, i.e. the types into which the selected component can be converted. Choose the target type and configure the properties that were not configured in the original component.

Previewing Output of Layout Definition Files

With IntelliJ IDEA, you can preview the output of the currently opened layout definition file without launching a physical or a virtual device. The preview changes dynamically as you update the layout definition file. Using the layout preview, you can adjust your application to different Android platforms, devices, orientations, dock modes, locales, etc.

Depending on the mode in which you are designing the application layout, you can preview the output in different ways:

- In the dedicated Preview tool window, when editing the layout definition file manually. The tool window appears when you open a layout definition file and switch to the Text tab.
- Right in the editor , when editing the layout using Android UI designer .

On this page:

- Accessing layout preview
- Previewing layout in different environments
- Adjusting the preview appearance

To access layout preview

1. Open the required layout definition file in the editor.

2. Choose the editing mode:
   - To edit the layout manually, switch to the Text tab. The Preview tool window opens.
   - To edit the layout in the Designer tool window, switch to the Design tab. In this tab you can edit the layout in the visual mode without editing the text definition file.

To preview a layout in different environments

Use the controls of the Preview tool window or the Design tab to emulate the target configuration to run the application in. The table below lists all available options:

Item	Tooltip	Description
	Configuration to render this layout in the IDE	From this drop-down list, select a layout configuration that you want to preview and edit, create a new layout configuration, or select different preview options. The available options are: – Create Landscape Variation: select this option to create a landscape version of your layout. The corresponding layout definition file will be generated in the `res\layout-land` folder. Once this variation is created, this menu option will be replaced with the Switch to layout-land option that opens the `layout-land\<layout_file_name>.xml` file for editing. – Create layout-xlarge Variation: select this option to create a variation of your layout for an extra large screen size (at least 960x720 dp). The corresponding layout definition file will be generated in the `res\layout-xlarge` folder. Once this variation is created, this menu option will be replaced with the Switch to layout-xlarge option that opens the `layout-xlarge\<layout_file_name>.xml` file for editing. – Switch to layout: this option is only available if you have created multiple layout versions. Select it to return to the original layout definition file. – Create Other: select this option to create another variation of your layout. In the Select Layout Directory dialog that opens, specify the folder where the layout definition will be stored and select resource qualifiers that determine a specific device configuration. Select the relevant qualifier and click ». Then specify the value of the qualifier in the dialog box that opens. The qualifier is added to the Chosen qualifiers list. – Preview Representative Sample select this option to display multiple device configurations and preview the layout on the most important screen sizes. – Preview All Screen Sizes: select this option to display multiple device configurations and preview the layout on all available screen sizes. – Preview All Locales: select this option to preview the layout in all locales where your application is going to be used. – Preview Right-to-Left Layout: select this option to preview the layout in both directions (left-to-right and right-to-left) side by side. – Preview Android Versions: select this option to preview the layout on all installed Android API versions. – Preview Included: select this option to preview your layout nested in another layout. This option is only available if the current layout is included into another layout. – Preview Layout Versions: select this option to display multiple device configurations and preview the layout in all available variations. – None: select this option to return to the default view. – Toggle Layout Mode: select this option to switch between different preview options.
Nexus 4	The virtual device to render the layout with	From this drop-down list, select a virtual or a physical device to preview what your application will look like on this device. To add a new virtual device, select Add Device Definition and configure an emulator in the Android Virtual Device (AVD) Manager that opens (for instructions refer to Managing Virtual Devices).
	Go to next state	From this drop-down list, select the preview orientation (portrait or landscape), the UI mode (Normal, Car Dock, Desk Dock, Television, Appliance) and switch between the Night (dimmed screen) and Not Night (standard brightness) modes. For details on UI modes refer to UiModeManager .
Holo	Theme	Click this button to select a theme from the Select Theme dialog.
	N/A	Click this button to associate the layout with an activity. Select Associate with <activity_name> to associate it with the current activity, or Associate with Other Activity to display a list of available activities to select from.
	Locale to render layout with in the IDE	From this drop-down list, select an existing locale or add a locale for your application. A locale is a combination of the target country and language to have the dates and some other data presented in accordance with the local rules and preferences. You can also preview the layout in all available locales and in both directions (left-to-right and right-to-left).
L	Android version to use when rendering layouts in the IDE	From this drop-down list, select an API version or use the Automatically Pick Best option to render the layout using the most suitable Android version. You can also preview the layout on all installed Android API versions.

Note Normally, the preview changes on-the-fly as you update the definition file. If it does not, click the Refresh button to have IntelliJ IDEA update the preview.

To adjust the preview appearance

When you preview the layout output in the Preview tool window and design the layout manually in the editor, you may need to move the bounds of the tool window so more space is available in the editor. In this case it may be helpful to compress the preview so it fits the tool window size but still reflects the layout output in the emulated environment.

Use the controls in the toolbar to adjust the appearance of the layout preview:

Item	Tooltip	Description
	Zoom to Fit	Toggle this button to have IntelliJ IDEA compress or expand the preview so it fits the target screen size .
	Reset Zoom to 100%	Click this button to have IntelliJ IDEA reset the zoom to preview the actual size.
	Zoom In	Click this button to have IntelliJ IDEA expand the preview.
	Zoom Out	Click this button to have IntelliJ IDEA compress the preview.
	Jump to Source	Click this button to switch to the Text tab where you can edit the application layout in the source `.xml` file.
	Refresh	Click this button to have IntelliJ IDEA update the preview so that it reflects on-the-fly changes to the current layout definition.
	Save Screenshot	Click this button to take a screenshot of the application preview. Note This button is only available from the Text tab of the layout preview window.
	Options	Click this button to configure the appearance and behavior of the tool window. – Hide for non-layout files: select this option to have IntelliJ IDEA temporarily close the tool window when the focus in the editor switches to a non-layout file. As soon as the same or another layout definition file is in focus, the tool window re-appears automatically. – Include Device Frames (if available): select this option to make the preview look like it is going to appear on the actual device. – Show Lighting Effect: select this option to display lighting effects to make the preview look more natural. Note This option is only available if the Include Device Frames option is selected.

Refactoring Android XML Layout Files

In addition to common refactoring , IntelliJ IDEA provides a number of the Android-specific refactorings for the Android layout definition XML files. Most of these refactorings are available in both design modes :

- The manual mode, right in the editor tab where the target layout definition file is opened, and from the Structure tool window .
- In the Android UI Designer , from the Design pane or Component Tree .

You can toggle between these modes by switching between the Design and Text tabs in the editor where the layout definition file is opened.

In this topic:

- Extract Style
- Inline Style
- Extract Layout
- Inline Layout Refactoring

Extract Style

Android styles help separate the application design from the application functionality, just like Cascading Style Sheets (CSS) .

With the extract style refactoring, you can create a style from a layout XML tag. IntelliJ IDEA creates a style declaration in the `/res/values/styles.xml` file and converts the tag attributes into style properties declared in the `item` elements. The benefit is that now you no longer need to parse an entire resource definition file updating the tag attributes to edit a layout. All you need to do is update the style definition, whereupon the changes are applied automatically wherever the style is referenced.

To apply the extract style refactoring

1. Do one of the following:
   - In the visual mode, select the component where the source XML tag is used in the definition on the canvas in the Design pane, or in the Component Tree.
   - In the manual mode, position the cursor anywhere inside the source XML tag.
2. From the main menu, select Refactor | Refactor This or press `Ctrl+Shift+Alt+T`. In the pop-up that appears, select Style from the Extract group. Alternatively, choose Refactor | Extract | Extract Style on the context menu of the selection.
3. In the Extract Android Style dialog box that opens, specify the name for the style to be created and select the tag attributes to be included as style properties.
4. To have IntelliJ IDEA automatically update the layout definition by replacing tag attributes with references to the newly created style, select the Launch 'Use Style Where Possible' refactoring after the style is extracted checkbox.

Inline Style

This refactoring is opposite to the Extract Style refactoring and results in adding all attributes defined in a style to one or all components where this style is applied. This refactoring is also used when you need to merge a parent style with its inheritor .

You can invoke the Inline Style refactoring from a style definition or from a component to which the style is applied. In the first case, IntelliJ IDEA removes the style definition and adds the corresponding attributes to all components where it is used. When refactoring is invoked from a component, you can apply it either to all components with this style, or only to the current one.

To apply the Inline Style refactoring

Do one of the following:

- To invoke the refactoring from a style definition:
  1. Open the style definition file, position the cursor at the name of the style to be inlined, and choose Refactor | Inline from the context menu or press `Ctrl+Alt+N`.
  2. The Inline Android Style dialog box that opens provides only one option (Inline all references and remove the style) which is selected by default. As a result, IntelliJ IDEA converts the attributes of the style into XML tags, adds them to the definitions of all components where the style is used, and removes the style definition.
     - To apply the changes immediately, click Refactor.
     - To open the Find tool window and preview the changes before applying , click Preview.
- To invoke the refactoring from a component definition:
  1. Select a component with the style in question:
     - In the visual mode, select a component with this style in the Design pane or in the Component Tree. Then choose Refactor | Inline Style from the main menu or from the context menu of the selection.
     - In the manual mode, select a component with this style in the Structure view and choose Refactor | Inline Style from the main menu or from the context menu of the selection.
  2. In the Inline Android Style dialog box that opens, configure the refactoring:
     - To have IntelliJ IDEA convert the attributes of the style into XML tags, add them to the definitions of all the components where the style is used, and remove the style definition, choose Inline all references and remove the style.
     - To have the refactoring applied to the current component only, choose Inline this usage and keep the style.
  3. Specify how you want to apply the changes.
     - To apply the changes immediately, click Refactor.
     - To open the Find tool window and preview the changes before applying , click Preview.

Extract Layout

IntelliJ IDEA supports re-using Android layouts : IntelliJ IDEA moves a part of an existing layout definition to a separate layout definition file and references it in the original layout definition through an automatically inserted `<include/>` tag. In IntelliJ IDEA, this operation is called extract layout refactoring.

To extract a layout

1. Select the fragment to extract as a separate layout. Do one of the following:
   - In the manual mode, select the code fragment to extract or position the cursor in the root element of the fragment.
   - In the visual mode, select the component or a group of components to be extracted in the Design. pane or in the Component Tree.
2. From the main menu, choose Refactor | Refactor This or press `Ctrl+Shift+Alt+T`. In the pop-up list that appears, choose Layout from the Extract group. Alternatively, choose Refactor | Extract | Extract Layout from the context menu of the selection.
3. In the Extract Android Layout dialog box that opens, specify the name of the file where the extracted layout definition will be stored and the name of its parent folder. You do not need to create the file and the folder in advance, IntelliJ IDEA will create them automatically.
4. Create a list of configuration qualifiers for the layout. This list determines the set of alternative resources that must be provided to enable running the application in specific target device configurations.
   - To add a qualifier to the set, select it in the Available Qualifiers list, click the » button, and specify the value of the qualifier.
   - To remove a qualifier from the set, select it in the Chosen Qualifiers list and click the « button.

   For detailed information on the qualifiers, their meaning, acceptable values and format, see specification of configuration qualifiers for alternative resources .

Inline Layout Refactoring

This refactoring is opposite to the Extract Layout refactoring and results in embedding a layout that was referenced through an `<include/>` tag and removing the tag.

The Inline Layout refactoring is invoked from the `<include/>` tag in the layout definition file. You can have IntelliJ IDEA apply it either to all references and remove the layout definition file, or only to the current reference.

To inline a layout

1. Open the XML definition file from which the layout in question is referenced.
2. Navigate to the `<include/>` tag which references the layout in question and position the cursor at the layout name.
3. Do one of the following:
   - Choose Refactor | Inline from the main menu.
   - Choose Refactor | Refactor This from the main menu, and then choose Inline in the pop-up list that opens,
   - Press `Ctrl+Alt+N`.
   - Choose Refactor | Inline from the context menu.
4. In the Inline Android Layout dialog box that opens, configure the refactoring:
   - To have IntelliJ IDEA embed the layout into all files where it is referenced through `<include/>` tags and remove the referenced layout definition file, select Inline all references and remove the file.
   - To have the refactoring applied to the current reference only, select Inline this usage and keep the file.
5. Specify how you want to apply the changes:
   - To apply the changes immediately, click Refactor.
   - To open the Find tool window and preview the changes before applying , click Preview.

Running and Debugging Android Applications

In this section

- Creating an Android Run/Debug Configuration
- Choosing the Target Device Manually
- Debugging with Logcat
- Managing Virtual Devices

On this page:

- Running or debugging an entire Android application
- Running or debugging a custom .apk that will be later embedded in an application
- Debugging an already running application

Note that you can also monitor an application behavior via the Android Device Monitor tool (to launch it, navigate to Tools | Android | Android Device Monitor).

Running or debugging an entire Android application

1. Start creating an Android run/debug configuration .
2. On the Run/Debug Configuration: Android page that opens, specify the configuration name and select the module to which this configuration will be applied.
3. Choose the Deploy default APK option in the Package area and Launch default Activity in the Activity area. IntelliJ IDEA will upload the `.apk` built from the module specified in the Module drop-down list above. The `.apk` is built automatically, no preliminary artifact configuration is required from your side.
4. Appoint the device where the application will run:
   - To use a virtual device, select the Emulator option in the Target Device area, choose an emulator from the drop-down list or create a new one .
   - To use a physical device, select the USB device option in the Target Device area and plug-in the device through a USB cable.
5. Start running or debugging the target activity. If you have not specified a target device, choose it manually .
6. View and analyze Android system messages in the Logcat tab of the Android tool window .

Running or debugging a custom .apk that will be later embedded in an application

1. Configure an artifact to generate the `.apk` from:
   - In the main menu, navigate to File | Project Structure .
   - In the left-hand pane, click Artifacts. In the central pane, click the Add button + in the toolbar.
   - Select Android Application from the list of available artifact types and then select Empty from the context menu.
   - In the right-hand pane, add the artifact components. The artifact must contain all resources and code that want packaged in the `.apk`. For details, refer to Generating a Signed Release APK Through an Artifact and Output Layout Tab .
2. Start creating an Android run/debug configuration .
3. On the Run/Debug Configuration: Android page that opens, specify the configuration name and select the module to which this configuration will be applied.
4. To run or debug a custom `.apk`, choose the Deploy custom artifact option in the Package area and select the artifact to build the `.apk` from. In this case, you have to define the artifact manually before creating a run/debug configuration (see Generating a Signed Release APK Through an Artifact and Working with Artifacts for details). Then select the Launch option in the Activity area and specify the start-up activity from the chosen artifact (`.apk`). Type the activity name manually or click the Browse button ... and select it in the Select Activity Class dialog box that opens.
   The list of available activities is determined by the choice of the module.
5. Appoint the device where the application will run:
   - To use a virtual device, select the Emulator option in the Target Device area, choose an emulator from the drop-down list or create a new one .
   - To use a physical device, select the USB device option in the Target Device area and plug-in the device through a USB cable.
6. Start running or debugging the target activity. If you have not specified a target device, choose it manually .
7. View and analyze Android system messages in the Logcat tab of the Android tool window .

Debugging an already running application

Apart from debugging an Android application by initiating a debugging session , you can apply the debugger to an already running application. You can do this in one of the following two ways:

- Attach the debugger to a running process:
  1. Click the Attach debugger to Android process button in the main toolbar.
  2. In the Choose Process dialog that opens, select a process from the list that shows all currently active processes grouped by the devices where they are running.
- Start a run/debug configuration without deploying a package and launching an activity
  1. Start creating an Android run/debug configuration . On the Run/Debug Configuration: Android page that opens, specify the configuration name and select the module to which this configuration will be applied.
  2. Select the Do not deploy anything option in the Package area and Do not launch Activity in the Activity area.
  3. Appoint the device where the application will run:
     - To use a virtual device, select the Emulator option in the Target Device area, choose an emulator from the drop-down list or create a new one .
     - To use a physical device, select the USB device option in the Target Device area and plug-in the device through a USB cable.
  4. Start running or debugging the target activity. If you have not specified a target device, choose it manually .
  5. View and analyze Android system messages in the Logcat tab of the Android tool window .

Creating an Android Run/Debug Configuration

For general information about run/debug configurations, see Creating and Editing Run/Debug Configurations .

To create an Android run/debug configuration

1. To start creating an Android run configuration, select Run | Edit Configuration from the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.

2. Click the Add New Configuration button + on the toolbar and select Android Application from the pop-up list.

3. On the Run/Debug Configuration: Android Application page that opens, specify the configuration name and select the module to which this configuration will be applied.

4. Specify the `.apk` file that will be deployed on the target device and appoint the activity that will be launched on the application start.
   - To run or debug the entire application, choose the Deploy default APK option in the Package area and Launch default Activity in the Activity area. IntelliJ IDEA will upload the `.apk` built from the module specified in the Module drop-down list above. The `.apk` is built automatically, no preliminary artifact configuration is required from your side.
   - To run or debug a custom `.apk` that will be later embedded in an application, choose the Deploy custom artifact option in the Package area and select the artifact to build the `.apk` from. In this case, you have to define the artifact manually before creating a run/debug configuration (see Generating a Signed Release APK Through an Artifact and Working with Artifacts for details). Then select the Launch option in the Activity area and specify the start-up activity from the chosen artifact (`.apk`). Type the activity name manually or click the Browse button ... and select it in the Select Activity Class dialog box that opens.
     The list of available activities is determined by the choice of the module.
   - If you are going to start a debugging session for an already running application, select Do not deploy anything in the Packages area to suppress uploading data to the device, and then select Do not launch activity in the Activity area. Executing a run configuration with these settings is the same as clicking the Attach debugger to Android process button on the toolbar (see Debugging a Running Application).

5. In the Target Device area, specify the device where the application will be launched.
   - To specify a virtual device, select the Emulator option and choose a virtual device from the Prefer Android Virtual Device drop-down list.

     Tip If no virtual devices are available, click the Browse button ... to start the Android Virtual Device (AVD) Manager and configure a new emulator (for details, refer to Managing Virtual Devices).
   - If you want to select a target device manually select the Show chooser dialog option. Each time you start a run/debug session and apply this configuration, IntelliJ IDEA will display the Choose Device dialog.
   - To have IntelliJ IDEA detect a plugged-in USB device upon the application start, select the USB device option.

   Selecting the Show chooser dialog or USB device option may be helpful if you are going to run the application on a physical device which will be plugged-in later and, therefore, the set of available devices cannot be foreseen.

Choosing the Target Device Manually

When you run or debug your Android application, and no target device is specified in the applied run/debug configuration, you need to choose the target device manually.

You have the following options:

- Choose a physical device
- Choose a running virtual device
- Configure a new virtual device

To choose a physical device

1. Plug-in a physical Android device through a USB cable.
2. Start a run/debug session .
3. In the Choose Device dialog that opens, select the Choose a running device option and choose the plugged-in device.

To choose a running virtual device

1. Start a run/debug session .
2. In the Choose Device dialog that opens, select the Launch emulator option and select a virtual device from the Android virtual device drop-down list.
   If you want to use this device emulator without selecting it manually each time you start a run/debug session, select the Use same device for future launches option.

To configure a new virtual device

1. Start a run/debug session .
2. In the Choose Device dialog that opens, select the Android virtual device option and click the Browse button ... to launch the Android Virtual Device (AVD) Manager . For instructions on how to configure a device emulator using the Android Virtual Device Manager, refer to Android documentation .

Debugging with Logcat

In IntelliJ IDEA, debugging of Android applications is provided through the support of the logcat functionality that stores a log of system debug output. Log messages include a stack trace when the emulator throws an error, so you can navigate to the exception location in the source code.

The logcat functionality is handled by the Android Debug Bridge (adb). This service supports interaction between your development environment, Android devices, emulators and other tools, for example, DDMS .

If various tools that use ADB are launched simultaneously, they may conflict with each other, so it is recommended to disable the logcat functionality before switching from IntelliJ IDEA to an ADB-managed tool.

In IntelliJ IDEA, the logcat functionality is available through the Logcat tab of the Android tool window . By default, the tab is activated automatically every time an application is deployed and launched successfully.

The logcat functionality is also available from the Debug tool window . The only difference is that while the Logcat tab in the Debug tool window shows a log of all processes, in the Android tool window you can filter log messages by a specific process.

Tip When logcat is attached to a device, it keeps auto-scrolling you to the bottom of the log. If you want to stop the automatic scroll, start scrolling the log with your mouse.

On this page:

- Switching the Logcat functionality on and off
- Showing and hiding the Logcat tab
- Defining the scope of log data to display

Switching the Logcat functionality on and off

- To enable/disable logcat, in the main menu navigate to Tools | Android and toggle the Enable ADB Integration option.

Showing and hiding the Logcat tab

By default, the Logcat tab is activated automatically every time an application is deployed and launched successfully. You can disable automatic display of the Logcat pane by performing the following steps:

1. Start creating an Android run/debug configuration or open an existing configuration for editing (in the main menu, navigate to Run | Edit Configurations and select the required configuration).
2. Switch to the Logcat tab and clear the Show logcat automatically check box.

Defining the scope of log data to display

During a debug session, you can switch to the Android tool window and configure the scope of log data to be displayed. You can:

- Select to only see messages related to a specific process.
- Define a log level for messages to be displayed.
- Create log data filter configurations.

For detailed instructions on how to configure these options, refer to Android tool window .

Managing Virtual Devices

Virtual devices are used to run or debug applications intended for Android devices. You can configure virtual devices to emulate actual Android units by defining their hardware and software parameters.

To configure a virtual Android device, you need to open the Android Virtual Device (AVD) Manager. To launch it:

- In the main menu, select Tools | Android | AVD Manager.
- When you create or edit a run/debug configuration, select the Emulator option in the Target Device area and click the Browse button next to Prefer Android Virtual Device.
- In the Choose Device Dialog that opens if you select the Show chooser dialog when creating a run/debug configuration, select the Launch emulator option and click the Browse button next to Android virtual device.

For instructions on how to configure Android virtual devices, refer to Android documentation.

Testing Android Applications

With IntelliJ IDEA you can generate and run unit tests for your Android applications using Android Testing Framework .

When the Android project is created, IntelliJ IDEA creates the following test directories containing sample tests:

- test directory - for running Unit tests
- androidTest directory - for running Instrumented Unit tests

To execute unit tests, IntelliJ IDEA provides an Android-specific run configuration.

To run tests for an Android application

1. Specify the tests that you want to run. Do one of the following:
   - To run a single test, open it in the editor and select Run | name of the test from the context menu. IntelliJ IDEA creates a temporary run configuration for your test.
   - To run all tests in a class or an entire test module, create an Android Test run/debug configuration .
2. Select a run/debug configuration from the drop-down list on the main toolbar and start the test according to it.
3. Monitor the test execution and view test results in the Test Runner tab of the Run tool window.

Sharing Android Source Code and Resources Using Library Projects

IntelliJ IDEA supports Android library projects that hold shared Android source code and resources. Other Android application projects can reference a library project and include its compiled sources in their `.apk` files at build time.

In IntelliJ IDEA, library projects are supported through separate library modules.

To enable sharing Android source code and resources, do one of the following:

- Create a new Library module.
- Convert an Application module that contains sources you want to share into a Library module.

In this topic:

- Creating a Library module
- Converting an Application module into a Library module
- Using a Library module in another project
- Adding data from AndroidManifest.xml for a library module to AndroidManifest.xml for the entire application
- Including the .dex file of a library module into the .apk of the entire application without rebuilding (pre-dexing)

Creating a Library module

1. Do one of the following:
   - Create a project from scratch
   - Add a module to an existing project
2. On the first page of the wizard, select Android in the left pane and Library Module in the right pane.
3. Complete the wizard.

Converting an Application module into a Library module

You can convert an Application module into a Library module by updating its Android facet.

1. Open the Project Structure dialog box by choosing File | Project Structure from the main menu.
2. Select Modules in the left pane. In the central pane, expand the node of the module that you want to turn into a Library module and click Android.
3. In the right-hand pane, select the Library module check box on top of the Android facet page .

Using a Library module in another project

To use a Library module in another project, you need to import this module into it.

1. From the main menu, select File | New | Module from Existing Sources.
2. In the dialog that opens , browse to the `.iml` module file that you want to import and click OK. The module node will be added to the tree view.
3. Add dependencies on the imported library module to the modules where its data is going to be used:
   1. Open the settings of the non-library module: navigate to File | Project Structure, select Modules in the left pane and select your non-library module in the central pane.
   2. In the right pane, switch to the Dependencies tab. Click the Add button + in the toolbar on the right and select Module Dependency from the context menu.
   3. In the Choose Modules dialog box that opens, select the imported library module from the list and click OK.

Adding data from AndroidManifest.xml for a library module to AndroidManifest.xml for the entire application

To successfully integrate a library module into another application, its components must be declared in the application `AndroidManifest.xml` file. The manifest file contains the information that is required to run the application (for more information, see App Manifest). You can either add this information on a library module manually, or extract it from `AndroidManifest.xml` of the library module and add it to the `AndroidManifest.xml` of the application automatically.

The second approach is referred to as merging manifests.

To have the manifest of a library module merged with the application manifest automatically:

1. Open the Project Structure dialog box by choosing File | Project Structure from the main menu.
2. Select Modules in the left pane. In the central pane, expand the non-library module and click the Android facet under its node.
3. In the right pane, on the Android facet page , switch to the Packaging tab and select the Enable manigest merging option.

Including the .dex file of a library module into the .apk of the entire application without rebuilding (pre-dexing)

During the application packaging, the `.class` files of a library module are converted into `.dex` files. This operation is referred to as dexing. Finally, the `.dex` files output from the library module is included in the final application `.apk` (learn more about the building procedure from Building and Running).

As a rule, the contents of a library module remain unchanged. In this case you can have them `dexed` only once, whereupon the output `.dex` files are included in the `.apk` . This approach is referred to as pre-dexing.

By default, IntelliJ IDEA pre-dexes library mode dependencies as well as external `jars` that have not been updated since the previous build. You can change these settings so that all `.class` files are always dexed.

1. Open the Project Structure dialog box by choosing File | Project Structure from the main menu.

2. Select Modules in the left pane. In the central pane, expand the non-library module and click the Android facet under its node.

3. In the right pane, on the Android facet page , switch to the Packaging tab and select the Pre-dex external jars and Android library dependencies option.

Renaming an Application Package (Application ID)

You may need to have your application built in several versions, which means that several Android application packages (`.apk` files) will be generated. If these files have the same name, the user will be unable to deploy them on the same device simultaneously. To avoid this, you can have IntelliJ IDEA generate several `.apk` files with different names (application IDs) from the same source code.

The application package name (application ID) is specified in the `package` attribute of the `manifest` element (see http://developer.android.com/guide/topics/manifest/manifest-element.html#package for details). This name follows the Java naming conventions and, by default, is the same as the name of the package to which the class implemented for the application belongs.

The name of the application (the `android:name` attribute of the `application` element, see http://developer.android.com/guide/topics/manifest/application-element.html#nm) and activity names (the `android:name` attribute of the `activity` element, see http://developer.android.com/guide/topics/manifest/activity-element.html#nm) are, by default, specified relative to the application ID and, accordingly, to the parent Java package of the application implementation class. However, renaming the application ID does not cause renaming the parent Java package of the application class.

You can either change the application ID through the Rename refactoring, or automatically on build time.

On this page:

- Changing an application ID through the Rename refactoring
- Renaming an application ID on build time

Changing an application ID through the Rename refactoring

1. Open the `AndroidManifest.xml` file.
2. Position the cursor at the `package` attribute of the `manifest` element and choose Refactor | Rename from the context menu.
3. In the Rename dialog box that opens, specify the new package name and click OK.

The value of the `package` attribute changes to the newly specified value. However, because this does not cause renaming the parent Java package of the application class, the relative names of the application and activities are replaced with their fully qualified names.

Renaming an application ID on build time

1. Open the Project Structure dialog box by choosing File | Project Structure from the main menu.
2. Select Modules in the left pane. In the central pane, expand the node of the relevant module and click Android.
3. In the right pane, switch to the Packaging tab and select the Rename manifest package option.

Generating Signed and Unsigned Android Application Packages

IntelliJ IDEA allows extracting Android application packages (.apk files). IntelliJ IDEA supports integration with the Android Asset Packaging Tool (aapt) which compiles the application resources (for detials, see Building and Running).

With IntelliJ IDEA, you can generate both signed and unsigned `.apk` files. The following options are available:

- Extract signed packages to deploy and run your applications on physical devices. Based on this signature, the Android system identifies the author of every deployed application. You do not need to apply for a personal signature to any authority, a signature generated by IntelliJ IDEA is quite sufficient. With IntelliJ IDEA, you can generate a signed package in one of the following two ways:
  - Use the Generate Signed APK Wizard . The package will be signed during extraction.
  - Configure the `.apk` file as an artifact by creating an artifact definition of the type Android application in the Release signed package mode. When IntelliJ IDEA builds the package in accordance with this definition, the package is signed automatically.
- Extract unsigned packages to test them on emulators. Unsigned packages can be extracted only through artifact definitions in the Release unsigned package mode.
- Extract and sign debug packages. This signature is sufficient for testing and debugging applications but does not allow publishing them. Signing packages in the debug mode is available only through configuring an artifact definition in the Debug package mode.

You can also have your application obfuscated during packaging through integration with the ProGuard built-in tool.

In this section:

- Generating a Signed Release APK Using a Wizard
- Generating a Signed Release APK Through an Artifact
- Generating an Unsigned Release APK
- Generating an APK in the Debug Mode
- Suppressing Compression of Resources

Generating a Signed Release APK Using a Wizard

To deploy and run an Android application on a physical device, you need to sign the application digitally. With IntelliJ IDEA, you can have your Android Application Package (`.apk` file) signed with an existing release key on package extraction. IntelliJ IDEA also incorporates a release key generation tool that can be invoked during the packaging procedure. Generated keys are saved in a keystore binary file.

You can have as many keystore files and keys as you need and use either existing keys, or create new ones in existing keystores, or even create new keystores.

If you use the Generate Signed APK Wizard , IntelliJ IDEA signs the package on extraction.

On this page:

- Extracting and signing an Android application package using a wizard
- Generating a new release key

Extracting and signing an Android application package using a wizard

1. From the main menu, select Build | Generate Signed APK. The Generate Signed APK Wizard starts.

2. On the first page , specify the release key you want to use and the keystore file that contains it. Do one of the following:
   - To sign the package with a key from an existing keystore file:
     1. Specify the file location in the Key store path text box. Type the path manually or click the Choose existing button and select the file in the dialog that opens . In the Key store password text box, type the password to the selected keystore.
     2. Specify the key alias and enter the password to access the key.
   - To generate a new key in an existing keystore:
     1. Specify the file location in the Key store path text box. Type the path manually or click the Choose existing button and select the file in the dialog that opens . In the Key store password text box, type the password to the selected keystore.
     2. Click the Create new button and configure the release key to be generated by filling in the data in the New Key Store dialog box that opens.
   - To generate a new keystore file with a new key:
     1. Click the Create new button. In the New Key Store dialog box that opens, specify the location of the file to be generated in the Key store path text box. Type the path manually or click the Browse button, then select the parent folder and specify the name of the file.
     2. Specify and confirm the password to access the keystore.
     3. Configure the new release key by filling in the data in the Key area.

   Click Next.

3. On the next step , in the Destination APK path text box, specify the folder where the `.apk` will be saved. Type the path manually, or click the Browse button and select the target folder in the dialog that opens.

4. To have IntelliJ IDEA obfuscate the application during the packaging procedure, select the Run ProGuard check box and specify the location of the proguard.txt configuration file. The file is generated on project creation and is stored in the project root. IntelliJ IDEA suggests this default location in the Config file path text box. Accept the suggestion or specify a custom configuration file by clicking the Browse button and selecting the required file in the dialog that opens .

5. Click Finish to generate and sign the package. IntelliJ IDEA informs you on successful operation completion.
   - To open the folder where the generated `.apk` file is located, click the Open File Location button.
   - To complete the wizard, click Close.

Generating a new release key

1. Open the New Key Store dialog box by doing one of the following:
   - On the first page of the Generate Signed APK wizard, click the Create new button.
   - When creating an Android Application artifact definition , click the Create new button in the Android tab.

2. Specify the keystore location and enter the password to access it.

   Note If you are going to add a new key to an already existing keystore and have already chosen it in the wizard or in the Artifact tab, these fields are already filled in.

3. Assign an alias to the new key in the Alias text box. The key will be referred to using this alias.

4. Enter the password to the key in the Password text box and confirm it.

5. Specify the validity period for the key in accordance with the expected lifespan of your application .

6. Provide the following personal information and click Next:
   - First and Last Name
   - Organizational Unit
   - Organization
   - City or Locality
   - State or Province
   - Country Code

Generating a Signed Release APK Through an Artifact

To deploy and run an Android application on a physical device, you need to sign the application digitally . With IntelliJ IDEA, you can have your Android Application Package (`.apk` file) signed with an existing release key on package extraction. IntelliJ IDEA also incorporates a release key generation tool that can be invoked during the packaging procedure. Generated keys are saved in a keystore binary file.

You can have as many keystore files and keys as you need and use either existing keys, or create new ones in existing keystores, or even create new keystores.

Apart from using the Generate Signed APK Wizard , you can configure the `.apk` file as an artifact by creating an Android application artifact definition. When IntelliJ IDEA builds the package in accordance with this definition, the package is signed automatically.

Extracting and signing a release Android application package using an artifact definition

1. Select File | Project structure from the main menu and click `Artifacts` in the left pane.
2. Click the New button + and select Android Application from the context menu.
3. In the popup menu, define the artifact contents by selecting one of the following options:
   - To create an empty layout definition, select Empty.
   - To include a module data in the artifact, select From module <module name>

   The general settings of the new artifact will be displayed in the Artifact Layout pane on the right.
4. Specify the general settings of the artifact. In the Output directory text box, specify the location of the target package `.apk` file.
5. Complete the artifact definition with the following steps:
   - Configure the artifact structure .
   - Add resources to the artifact.
   - Arrange the elements included in the artifact.
   - If necessary, specify additional activities to be performed before and after building the artifact in the Pre-processing and Post-Processing tabs.
6. Switch to the Android tab tab and select Release signed from the Type drop-down list.
7. Specify the release key you want to use and the keystore file that contains it. Do one of the following:
   - To sign the package with a key from an existing keystore file:
     1. Specify the file location in the Key store path text box. Type the path manually or click the Choose existing button and select the file in the dialog that opens . In the Key store password text box, type the password to the selected keystore.
     2. Specify the key alias and enter the password to access the key.
   - To generate a new key in an existing keystore:
     1. Specify the file location in the Key store path text box. Type the path manually or click the Choose existing button and select the file in the dialog that opens . In the Key store password text box, type the password to the selected keystore.
     2. Click the Create new button and configure the release key to be generated by filling in the data in the New Key Store dialog box that opens.
   - To generate a new keystore file with a new key:
     1. Click the Create new button. In the New Key Store dialog box that opens, specify the location of the file to be generated in the Key store path text box. Type the path manually or click the Browse button ..., then select the parent folder and specify the name of the file.
     2. Specify and confirm the password to access the keystore.
     3. Configure the new release key by filling in the data in the Key area.

   Click Next.
8. To have IntelliJ IDEA obfuscate the application during packaging, select the Run ProGuard check box and specify the location of the proguard.txt configuration file. The file is generated on project creation and is stored in the project root. IntelliJ IDEA suggests this default location in the Config file path text box. Accept the suggestion or specify a custom configuration file by clicking the Browse button ... and selecting the required file in the dialog that opens .

Generating an Unsigned Release APK

Running an Android application on an emulator does not require a digital signature, so you can use unsigned packages for this purpose. An unsigned package can be extracted only through an Android artifact definition with the Release unsigned package mode turned on.

Extracting an unsigned release Android application package

1. Select File | Project structure from the main menu and click `Artifacts` in the left pane.
2. Click the New button + and select Android Application from the context menu.
3. In the popup menu, define the artifact contents by selecting one of the folloiwng options:
   - To create an empty layout definition, select Empty.
   - To include a module data in the artifact, select From module <module name>

   The general settings of the new artifact will be displayed in the Artifact Layout pane on the right.
4. Specify the general settings of the artifact. In the Output directory text box, specify the location of the target package `.apk` file.
5. Complete the artifact definition with the following steps:
   - Configure the artifact structure .
   - Add resources to the artifact.
   - Arrange the elements included in the artifact.
   - If necessary, specify additional activities to be performed before and after building the artifact in the Pre-processing and Post-Processing tabs.
6. Switch to the Android tab tab and select Release unsigned from the Type drop-down list.
7. To have IntelliJ IDEA obfuscate the application during packaging, select the Run ProGuard check box and specify the location of the proguard.txt configuration file. The file is generated on project creation and is stored in the project root. IntelliJ IDEA suggests this default location in the Config file path text box. Accept the suggestion or specify a custom configuration file by clicking the Browse button ... and selecting the required file in the dialog that opens .

Generating an APK in the Debug Mode

Apart from generating signed packages to be deployed on physical devices, or unsigned packages to be run on emulators, you can also have an application package extracted and signed in the debug mode . This signature is sufficient for testing and debugging applications, but does not allow publishing them. Signing an application package in the debug mode is available only through configuring an artifact.

In the debug mode, you can have an APK signed either with the default certificate or with a custom one.

If you decide to use the default certificate, IntelliJ IDEA signs the extracted package in the debug mode using the debug keystore or a key that is generated by the Android SDK tools and has predefined names and passwords:

- Keystore name: `debug.keystore`
- Keystore password: `android`
- Key alias: `androiddebugkey`
- Key password: `android`
- CN (common name): `CN=Android Debug,O=Android,C=US`

These are the default settings in IntelliJ IDEA. This means that if you do not configure an artifact manually and select the Deploy default APK option in the Run/Debug Configuration: Android Application dialog box , IntelliJ IDEA will use the predefined values in the certificate for the generated `.apk` .

If you use a custom certificate, IntelliJ IDEA signs the extracted package in the debug mode using the debug keystore or a key that you specify yourself . You can have a new certificate generated or reuse an existing one. The latter approach is helpful, for example, if you have several applications and you want them all signed with he same certificate so they can be stored in the same folder on the device.

Signing a package in the debug mode

1. Select File | Project structure from the main menu and click `Artifacts` in the left pane.
2. Click the New button + and select Android Application from the context menu.
3. In the popup menu, define the artifact contents by selecting one of the folloiwng options:
   - To create an empty layout definition, select Empty.
   - To include a module data in the artifact, select From module <module name>

   The general settings of the new artifact will be displayed in the Artifact Layout pane on the right.
4. Specify the general settings of the artifact. In the Output directory text box, specify the location of the target package `.apk` file.
5. Complete the artifact definition with the following steps:
   - Configure the artifact structure .
   - Add resources to the artifact.
   - Arrange the elements included in the artifact.
   - If necessary, specify additional activities to be performed before and after building the artifact in the Pre-processing and Post-Processing tabs.
6. Switch to the Android tab and specify the certificate you want to use from the Type drop-down list:
   - Select Debug signed with default certificate to have IntelliJ IDEA use the debug keystore or a key that is generated by the Android SDK tools and has predefined names and passwords.
   - Select Debug signed with custom certificate to have the package signed with a certificate of your choice. Specify the key to use and the keystore file that contains it by doing one of the following:
     - To sign the package with a key from an existing keystore file:
       1. Specify the file location in the Key store path text box. Type the path manually or click the Choose existing button and select the file in the dialog that opens . In the Key store password text box, type the password to the selected keystore.
       2. Specify the key alias and enter the password to access the key.
     - To generate a new key in an existing keystore:
       1. Specify the file location in the Key store path text box. Type the path manually or click the Choose existing button and select the file in the dialog that opens . In the Key store password text box, type the password to the selected keystore.
       2. Click the Create new button and configure the release key to be generated by filling in the data in the New Key Store dialog box that opens.
     - To generate a new keystore file with a new key:
       1. Click the Create new button. In the New Key Store dialog box that opens, specify the location of the file to be generated in the Key store path text box. Type the path manually or click the Browse button ..., then select the parent folder and specify the name of the file.
       2. Specify and confirm the password to access the keystore.
       3. Configure the new release key by filling in the data in the Key area.

   Click Next.
7. To have IntelliJ IDEA obfuscate the application during packaging, select the Run ProGuard check box and specify the location of the proguard.txt configuration file. The file is generated on project creation and is stored in the project root. IntelliJ IDEA suggests this default location in the Config file path text box. Accept the suggestion or specify a custom configuration file by clicking the Browse button ... and selecting the required file in the dialog that opens .

Suppressing Compression of Resources

By default, the Android Asset Packaging Tool (aapt) compresses resources during packaging. However you can have resources of a certain type included in the `.apk` file uncompressed by changing the settings of the Android facet .

To suppress compressing resources of a certain type

1. Select File | Project structure from the main menu.
2. In the left-hand pane, select Modules, then in the central pane click the Android facet under the module with the resources to be packaged uncompressed.
3. In the right-hand pane, switch to the Compiler tab.
4. In the Additional command line parameters text box, type `-0 <file extension for this type of resources>`. As a result, all files with the specified extension will be excluded from compression. If the set of additional parameters does not fit into the text box, click and specify the parameters in the dialog that opens.
   For example, if you want to include resources of a certain type in an uncompressed format, type `-0 <file extension for this type of resources>`.

Accessing Android SQLite Databases from IntelliJ IDEA

If your application uses an Android SQLite database, you can access this database right from IntelliJ IDEA through a data source of the Android SQLite type. For more information about IntelliJ IDEA data sources, see Managing Data Sources .

Creating an Android SQLite data source

1. Open the Database tool window by selecting View | Tool Windows | Database from the main menu.
2. Click + on the toolbar and select Android SQLite from the drop-down menu.
3. In the Data Sources and Drivers dialog that opens, specify the following:
   - The name of the data source.
   - The physical or virtual device where the target database is stored. If no devices are available in the drop-down list, this means that there are no running devices connected to IntelliJ IDEA. Run and connect a physical device or launch an emulator.
   - Specify the name of the application package the target database is associated with. For more information about application packages, see Working with AndroidManifest and Android documentation . Select a package name suggested by IntelliJ IDEA or type its ID.
     For the database to be accessible, the corresponding application must be built as debuggable and installed on the device or the emulator.

     IntelliJ IDEA run configurations, by default, build Android applications in the debug mode. Alternatively, you can generate the APK in the debug mode .
   - In the Storage area, specify where the database is located:
     - Choose Internal if the database is stored in the internal memory of the device or the emulator.
     - Choose External if the database is stored in the external memory of the device or the emulator.
   - In the Database drop-down list, specify the database name or location:
     - If the database is stored in the internal memory, choose the database name.
     - If the database is stored in the external memory, specify the database location relative to the memory root. For example, `Android/data/<application_ID>/<database_name>` .
   - If the necessary SQLite driver files are missing, download them by clicking the Download link at the bottom of the dialog box.

See also, Managing Data Sources .

Android Tutorials

An Android application is a Java program written against the Java SDK and Android SDK. An integrated development environment (IDE) for any kind of Java applications, IntelliJ IDEA automates and streamlines all steps of an Android application development, from writing the source code to preparing your application for publishing.

IntelliJ IDEA helps you perform the following tasks:

- Create the skeleton of a fully-functional Android application
- Manage the project and add classes and resources, such as strings, layouts, graphics, etc.
- Preview changes to the user interface through a tailor-made graphical designer
- Write and debug your source code
- Create unit tests
- Package and run your application on both physical devices and emulators

These tutorials will help you set up an Android project in IntelliJ IDEA and produce an executable that can run on emulators and physical devices.

Note that the tutorials listed below imply that you are using Windows as your operating system. Instructions for other operating systems may vary slightly.

In this section:

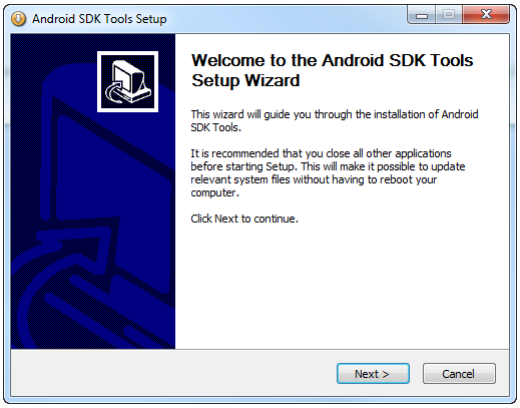
- Prerequisites for Android Development
- Creating a New Android Project
- Importing an Existing Android Project
- Exploring the Project Structure
- Building and Running the Application
- Editing UI Layout Using Designer
- Editing UI Layout Using Text Editor
- Making the Application Interactive
- Creating Unit Tests
- Packaging the Application

Prerequisites for Android Development

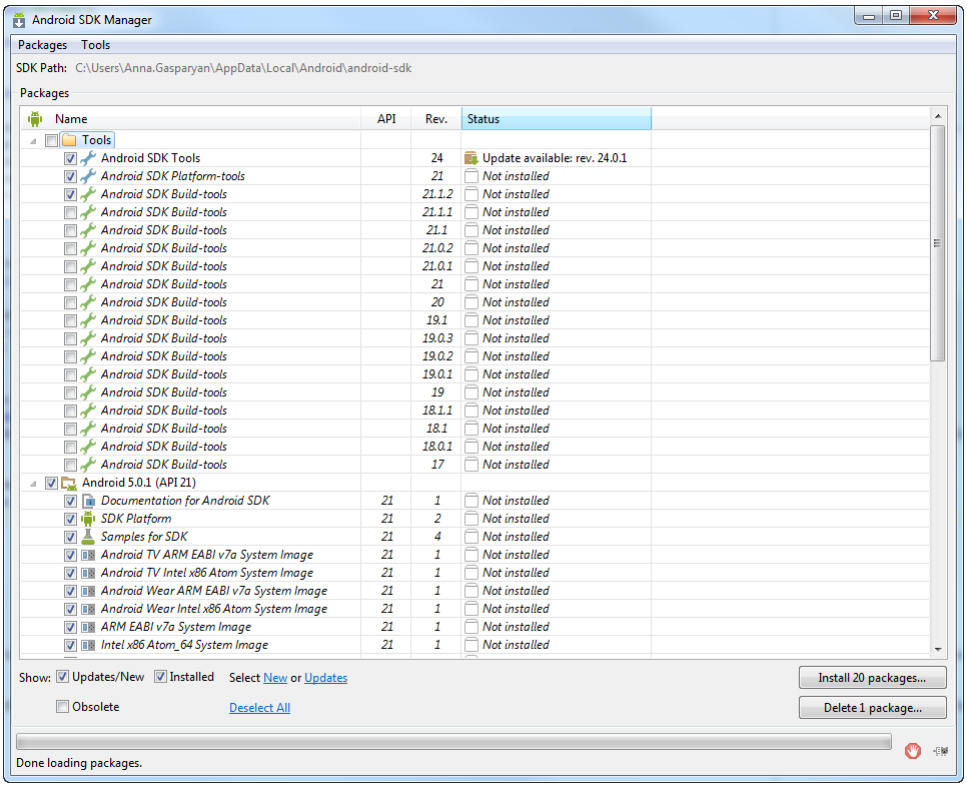
Before you start writing your first "Hello, world" Android application in IntelliJ IDEA, do the following:

1. Download and install JDK 7.
2. Download the Android SDK tools.

   Note that the official Android download page provides different download options. The default download option is Android Studio that includes the Android SDK tools. Since you already have IntelliJ IDEA with a bundled Android plugin, you can choose a smaller download and just pick up the Android SDK and platform tools. Scroll to Other Download Options and select the SDK Tools Only package for your operating system.
3. Run the installer to launch the Android SDK Tools Setup wizard:



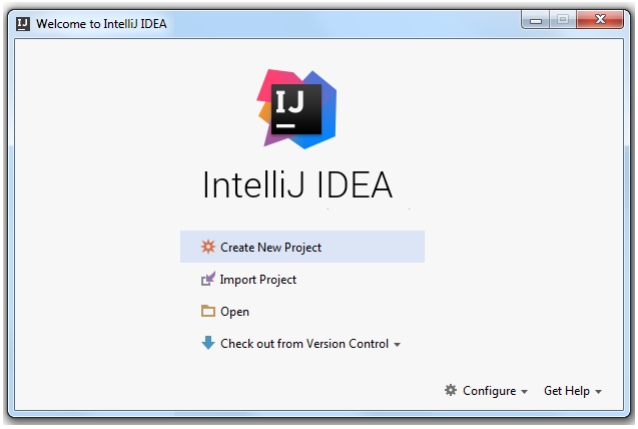
   First, the wizard checks if the Java SDK is installed. Next, choose the installation options and specify the destination folder. We strongly recommend that the path to the Android SDK home directory does not contain spaces.
4. After the installation has completed successfully, you need to add SDK packages using the SDK Manager. On the last page of the Android SDK Tools Setup wizard, select the Start SDK Manager option and click Finish.
5. In the Android SDK Manager, select the packages you want to install (for detailed instructions, refer to Adding SDK Packages).



   For the procedures covered in our tutorials, you can leave the default selections. You can always launch the SDK Manager from the installation folder if you need to install more packages.

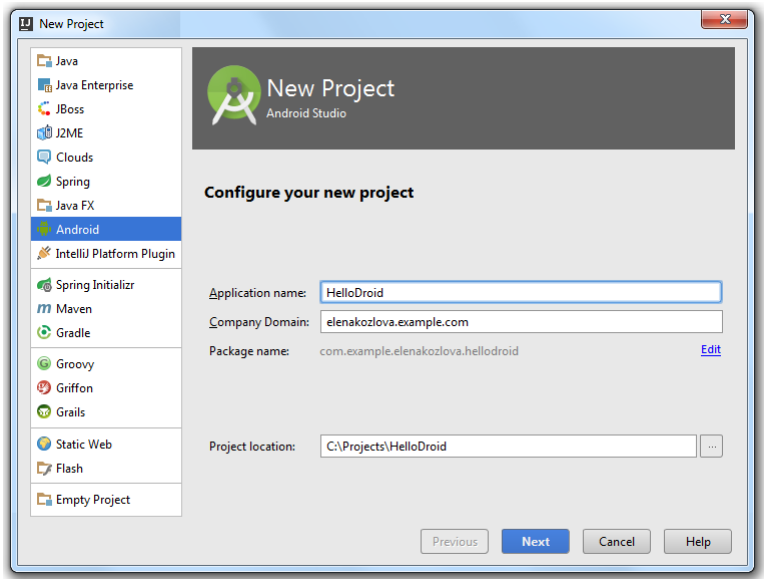
Creating a New Android Project

1. Launch the New Project wizard . If no project is currently opened in IntelliJ IDEA, click Create New Project on the Welcome screen:



Otherwise, select File | New | Project from the main menu.

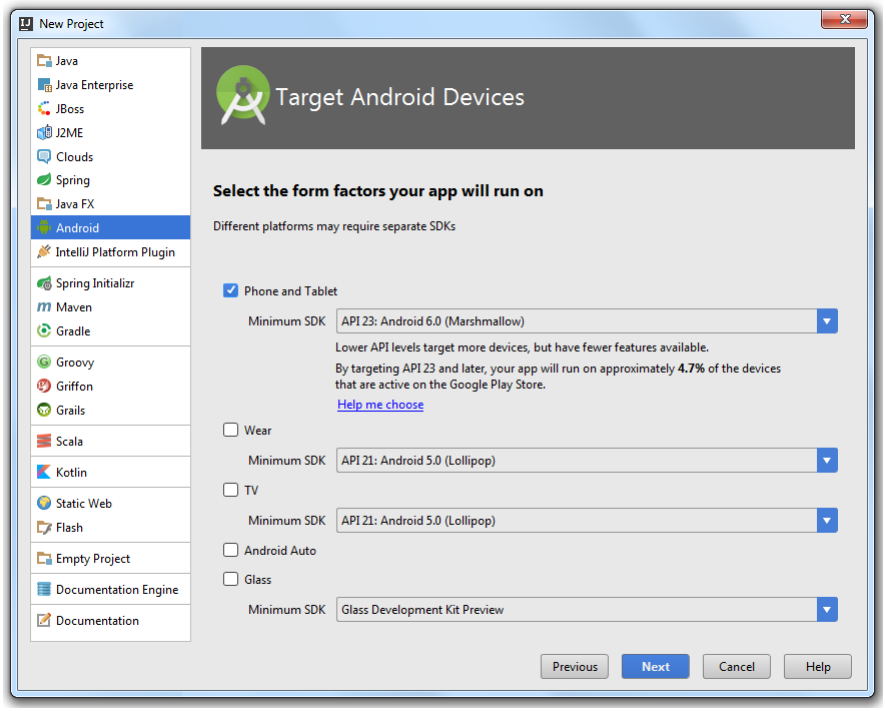
2. On the first page of the wizard, select Android in the left pane. In the right pane, IntelliJ IDEA automatically creates the name of your application, company domain, package name and the project location:



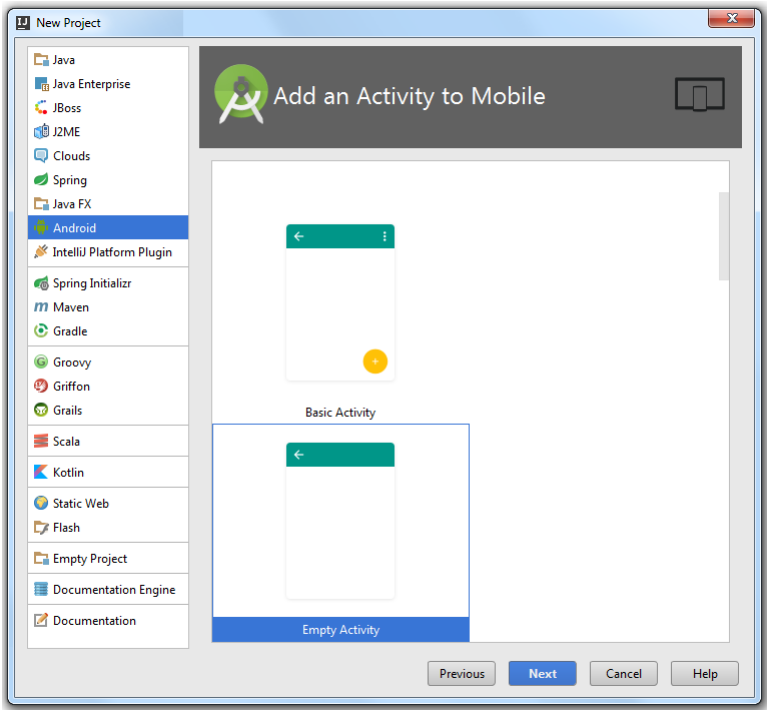
You can edit the specified fields if you like. In our case, we have changed the default name of our application to `HelloDroid` .

Note that the package name must have the following format: com.xxx.yyy, where xxx usually stands for your company name, and yyy is the application name. Note that you can use any names here, but the suggested pattern significantly reduces the risk of name conflicts with other applications.

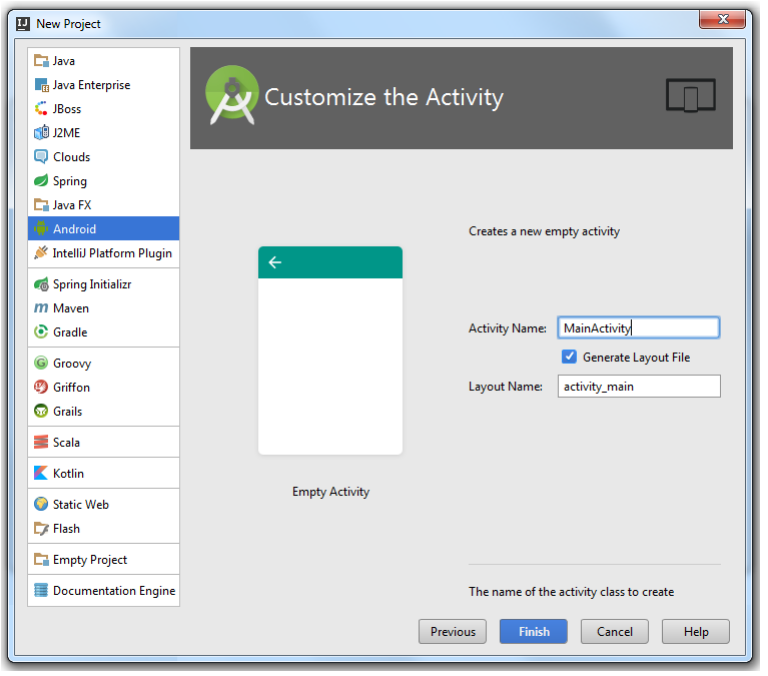
3. On the next page of the wizard, select a target device where the application will be run and debugged.



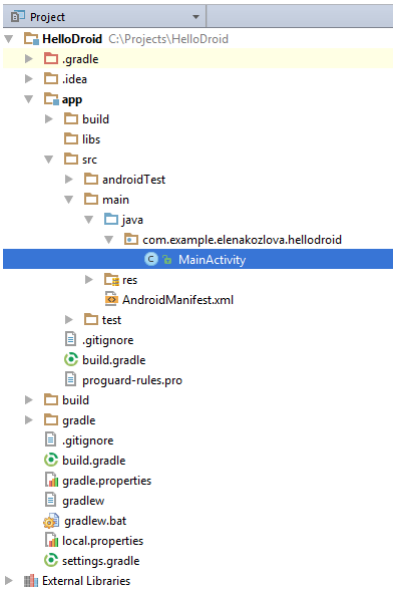
4. On the next page of the wizard, add an activity for your module.



5. On the next page of the wizard, customize the selected activity. Click Finish to complete the wizard.



6. Your Android project will be created with the predefined project structure:

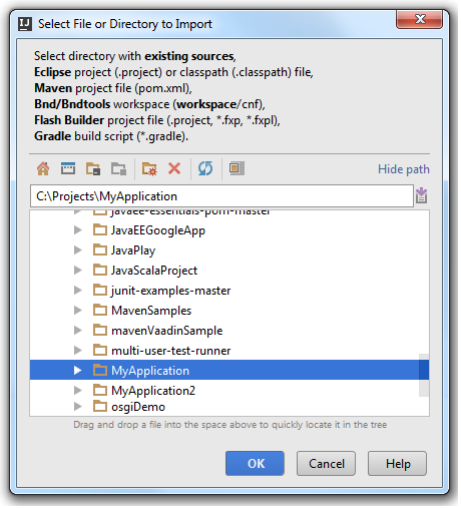


Importing an Existing Android Project

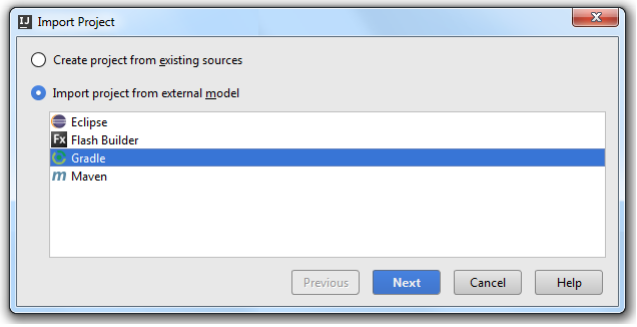
With IntelliJ IDEA you can not only create an Android project from scratch, but also import an existing project developed using other tools. The default scenario is to import an existing Android-Gradle project . However, you can also import a Maven , Eclipse or Flash Builder project, or even build a new project from a bunch of source files.

As a rule, when you import a project, your source files remain in their original location - IntelliJ IDEA simply creates a superstructure that allows treating your sources, libraries and other assets as an IntelliJ IDEA project. IntelliJ IDEA automatically generates the `.idea` directory that contains a set of configuration files (`.xml`), and a project file (`.iml`) for each of the project modules (for more details, see Project and Module).

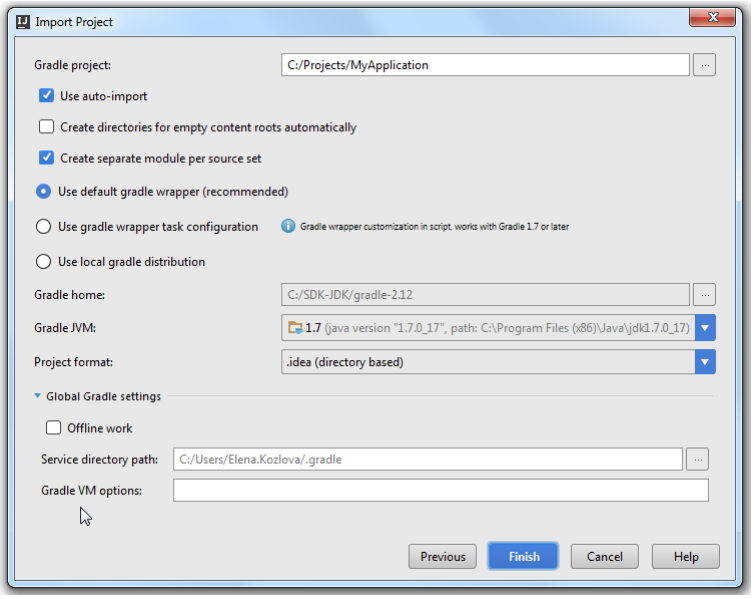
1. Select File | New | Project from Existing Sources from the main menu.
2. In the dialog that opens, browse to the project (or a directory containing source files) that you want to import:



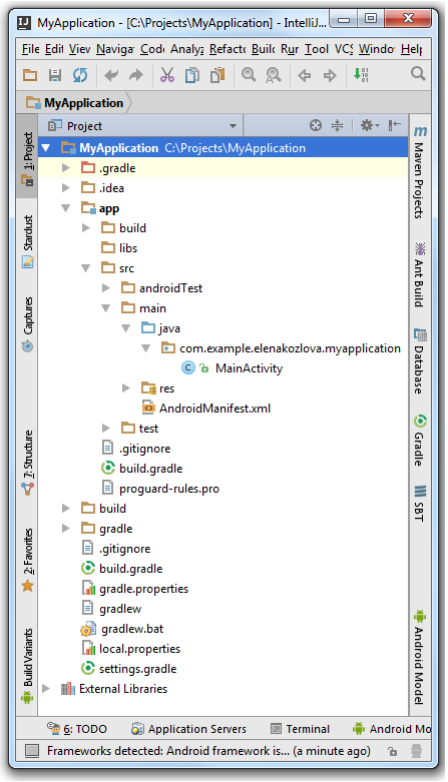
3. On the second step, select whether you want to build a project from scratch using the files under the specified directory, or whether you want IntelliJ IDEA to build a project according to the selected model and proceed in a more automated way. In our case IntelliJ IDEA is aware that this Android project uses Gradle build system, recognizes the `build.gradle` file and suggests importing from Gradle.



4. On the third step, you can modify Gradle settings and click Finish.



The newly created IntelliJ IDEA project built from the external Gradle model will open.


MyApplication - [C:\Projects\MyApplication] - IntelliJ...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyApplication
Project
MyApplication C:\Projects\MyApplication
.gradle
.idea
app
build
libs
src
androidTest
main
java
com.example.elenakozlova.myapplication
MainActivity
res
AndroidManifest.xml
test
.gitignore
build.gradle
proguard-rules.pro
build
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle
External Libraries
1: Project
Stardust
Captures
7: Structure
2: Favorites
Build Variants
Maven Projects
Ant Build
Database
Gradle
SBT
Android Model
6: TODO
Application Servers
Terminal
Android Mo
Frameworks detected: Android framework is... (a minute ago)

Exploring the Project Structure

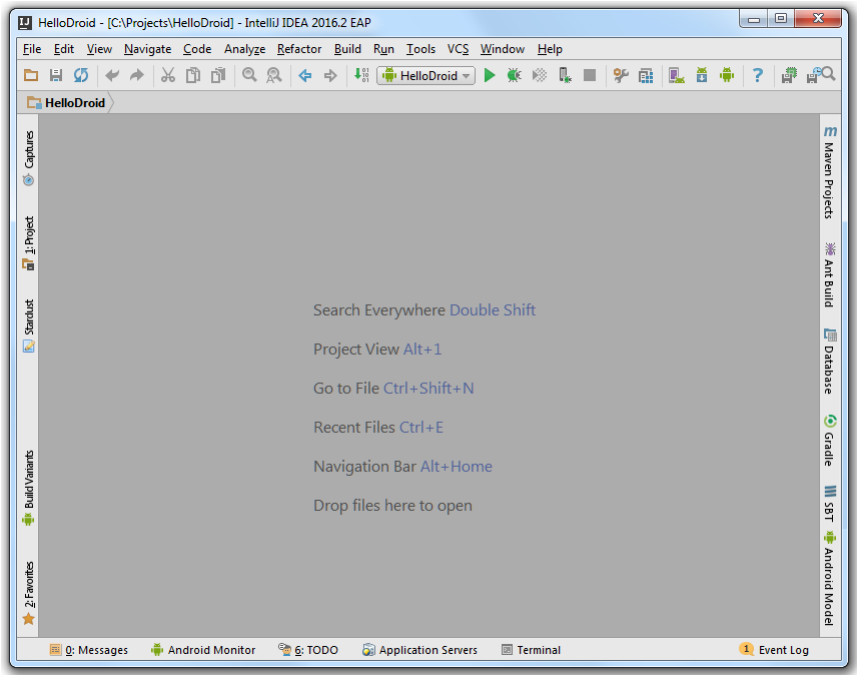
An Android project is primarily a Java project, so it contains many of the standard folders you find in a Java project. Let's find out more.

In this tutorial:

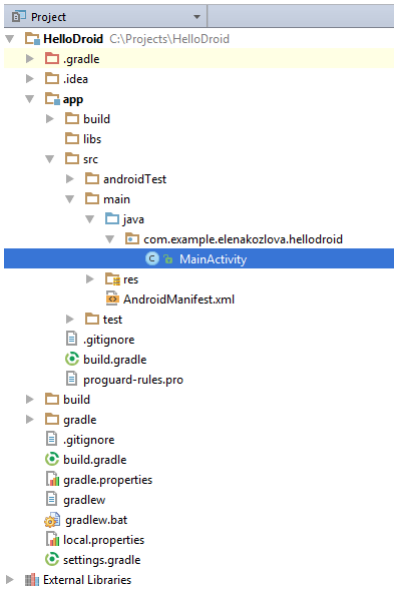
– Open the Project View
– Explore Code-Related Folders
– Explore the Resource Folders
– Explore the AndroidManifest.xml File

Open the Project View

If the project view is collapsed to the left edge, click Project in the left gutter, or just press `Alt+1`.



When you expand the Project tool window, it shows the typical structure of an Android project that is analogous to the structure of most Java projects:



Explore Code-Related Folders

An Android project contains the following key folders and files:

– .idea folder: contains a few subfolders and various XML files that mostly contain internal IntelliJ IDEA information and general settings. Normally, there is no need to edit the contents of this folder.

– src folder: contains all source files that make up the application (activities, helper classes, etc.). You can build any hierarchy of subfolders under the `src` folder to better reflect the structure and the complexity of your application.

– res folder: contains all project resources, such as drawable resources, layouts, etc. (for more details, see Explore the Resource Folders).

– libs folder: contains all class libraries (`.jar` files) that you want to reference from the source files of your application. You can simply drag-and-drop `.jar` files from the disk into this folder.

Explore the Resource Folders

The `res` folder contains all external resources used by your application, such as image assets, layout files, strings, menus, etc. Most resources (except for images) are expressed through `.xml` files. The `res` folder usually contains the following subfolders:

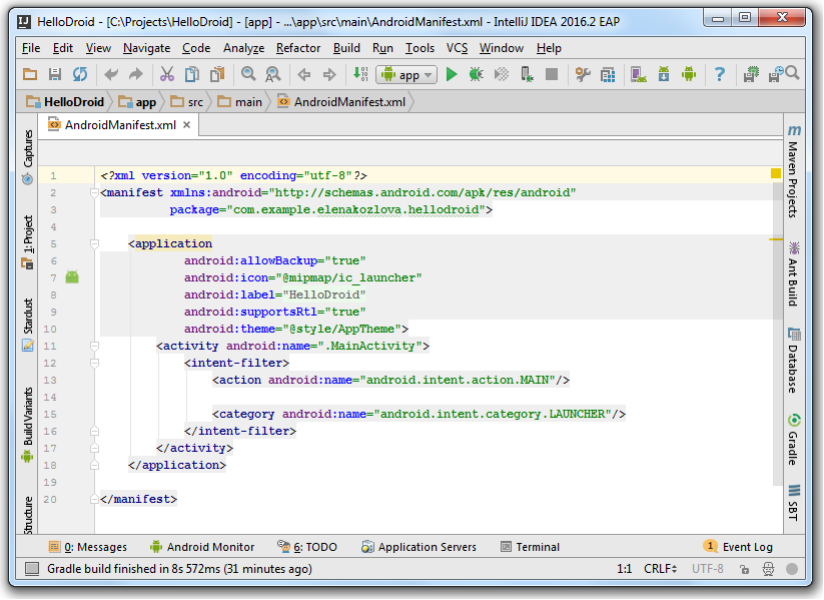
– drawable: contains all images you reference from within the application. There are actually four different drawable folders designed to contain images in different resolutions for devices with different screen density (expressed as `dpi` - dots per inch):
  – hdpi (high density, 240)
  – ldpi (low density, 120)
  – mdpi (medium density, 160)
  – xhdpi (extra high density, 320)

  Designing drawable resources in different resolutions is required to support different devices and multiple screens (for more details, see to Supporting Multiple Screens).
– layout: contains layout files (`.xml`) used to define the user interface of an activity or an application widget. You can edit layout definition files manually, or through the integrated graphical designer (for more details, see Designing Layout of Android Application).
– values: contains `.xml` files that declare strings, graphical styles, and colors. Normally, data stored in these files is expressed in the form of name/value pairs.
– menu: contains definitions of menus to be used by the application. The `menu` folder is not generated when you create a new project, and only appears in the project structure after you have created the first menu.

Explore the AndroidManifest.xml File

Each Android application must have the `AndroidManifest` file in its root directory. This file contains general information about the application processed by the Android operating system. This information is essential to run the application.

Among other things, the `AndroidManifest.xml` file declares the package name (that serves as a unique identifier for your application), and the minimal version of the Android SDK required for the device where the application will run. It also declares the entry point in the code for the operating system to launch the application, along with permissions the application requires. For more details on the `AndroidManifest` file, see App Manifest .



Building and Running the Application

After you have created your first Android application, you are ready to compile it and deploy it to an Android emulator, or a physical device.

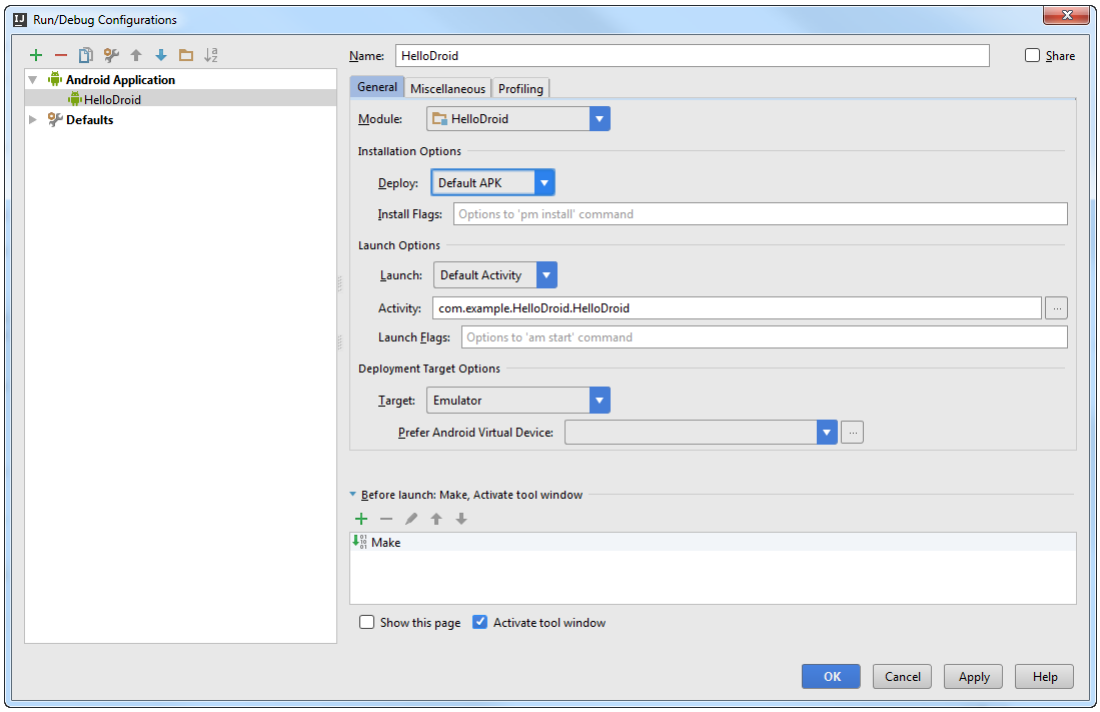
Before you build the project and start testing the application, it is recommended that you review the current build configurations.

In this tutorial:

- Review the default run/debug configuration
- Add a New Run/Debug Configuration
- Test the Application on an Emulator
- Test the Application on a Physical Device
- Debug your Application
- Analyze Debug Output

Review the default run/debug configuration

Click Run | Edit Configurations. The following dialog will be displayed:

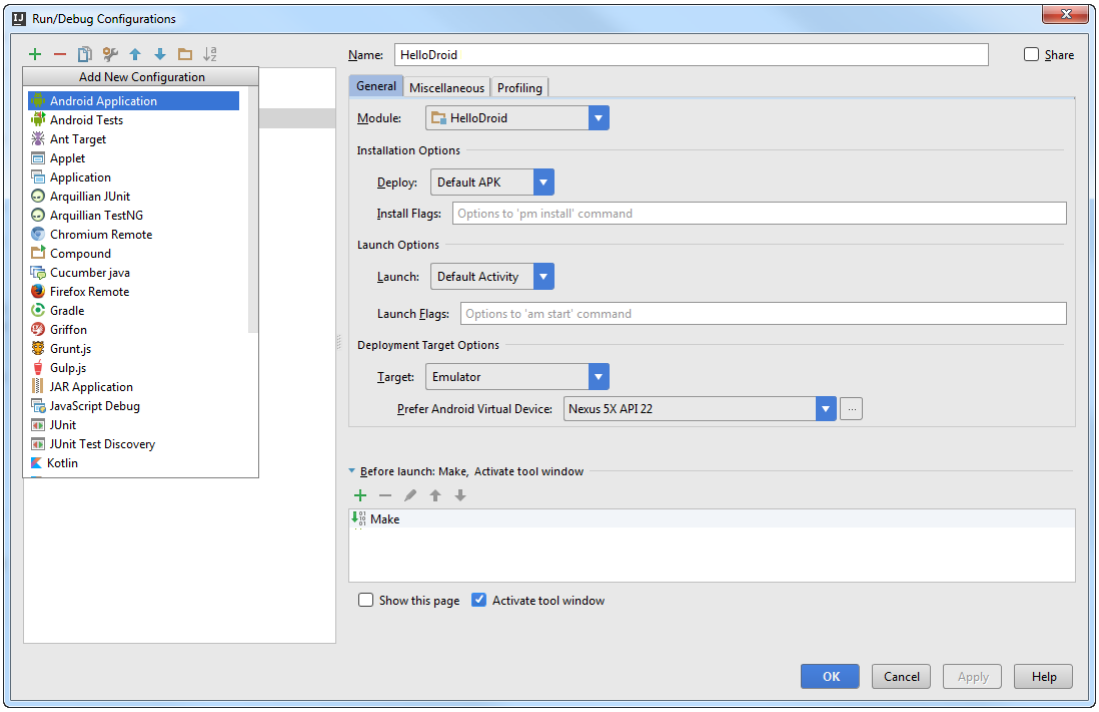


IntelliJ IDEA automatically creates a default run/debug configuration with the same name as your project. For a detailed description of each configuration option in an Android run/debug configuration, see Run/Debug Configuration: Android Application .

Add a New Run/Debug Configuration

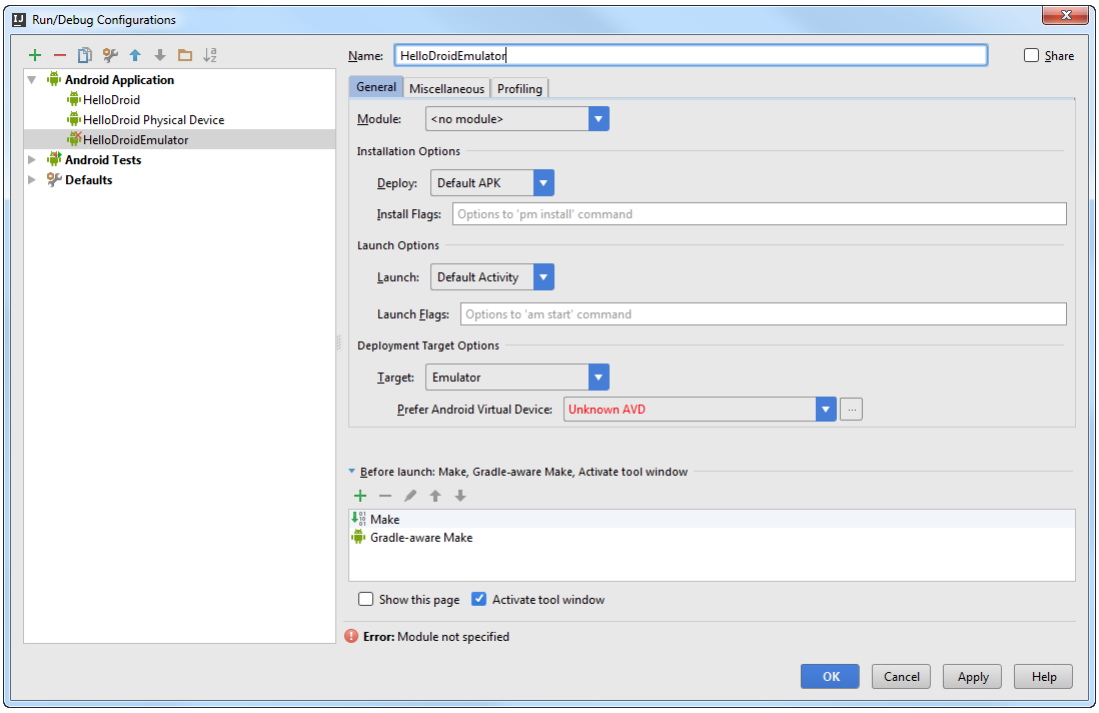
Let's create a custom build configuration:

1. In the Run/Debug Configurations dialog, click the Add button + and select Android Application from the list:



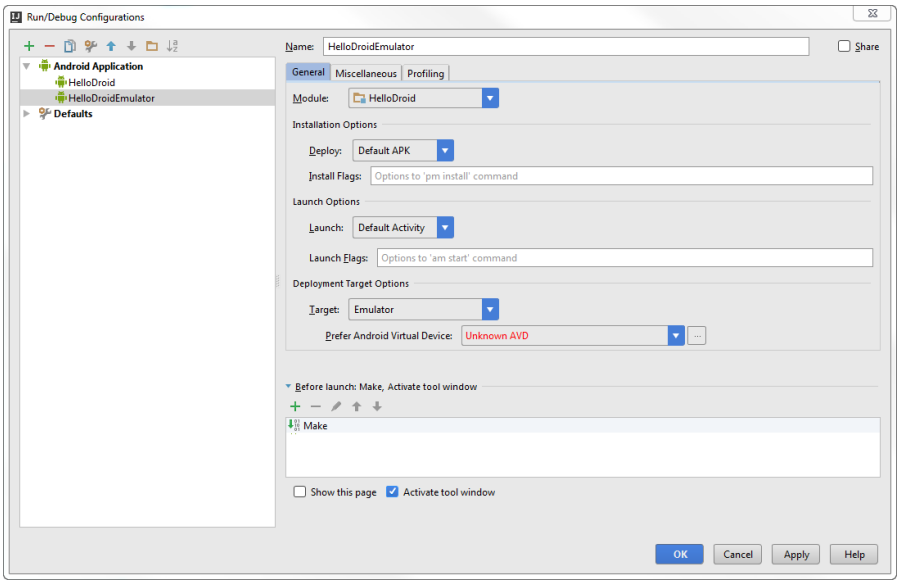
2. A new item will be added to the tree in the left pane with the default name `Unnamed` . Select it and enter a more meaningful name in the Name field, for example,

`HelloDroid Emulator`:



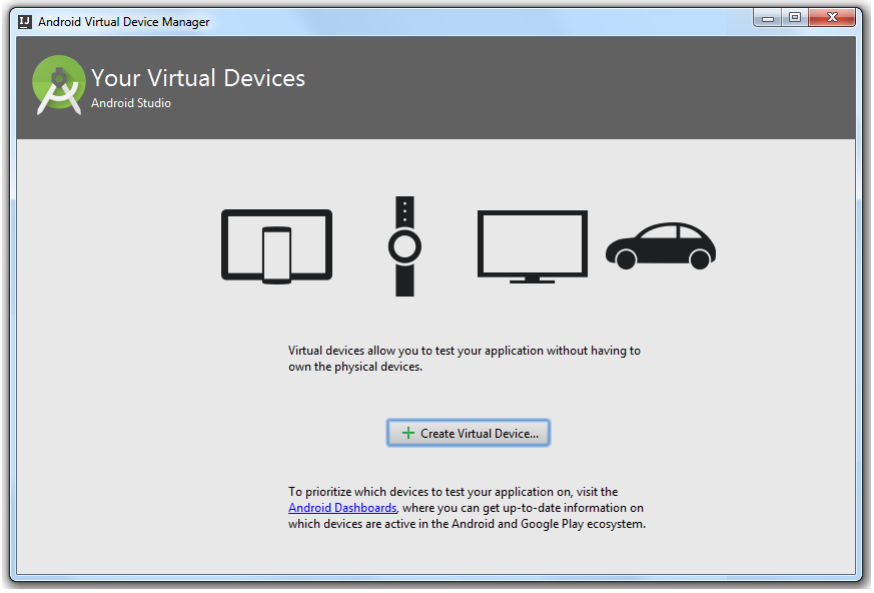
Note that the newly created configuration is shown with a red cross to indicate that it is not yet ready to be used, as no module has been specified.

3. Next, you need to specify the executable module. Expand the Module drop-down list and select `app`: it is the only executable module in your project.

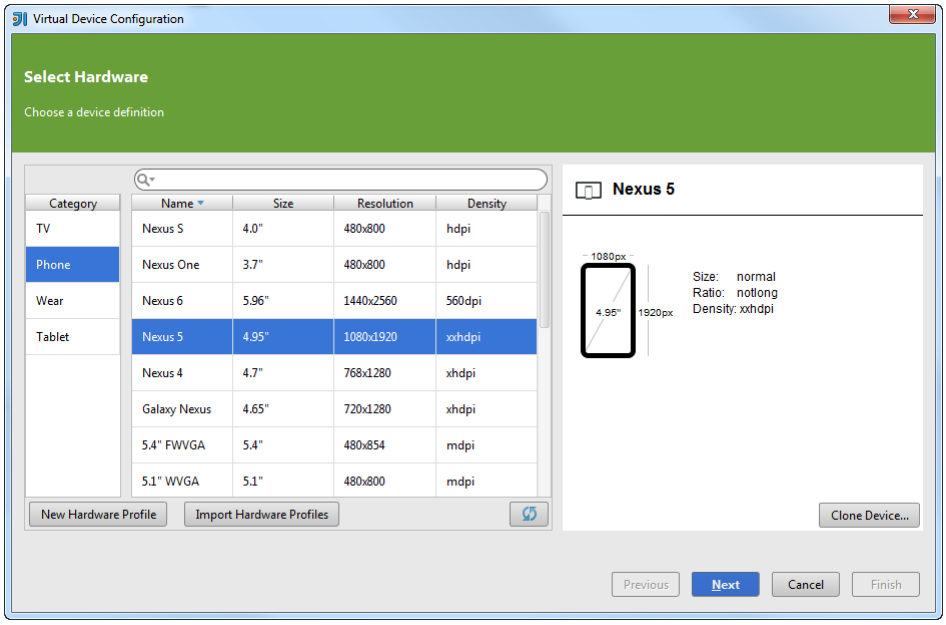


When you have selected a module, the red cross disappears, as now your run/debug configuration is filled with the minimum required information.

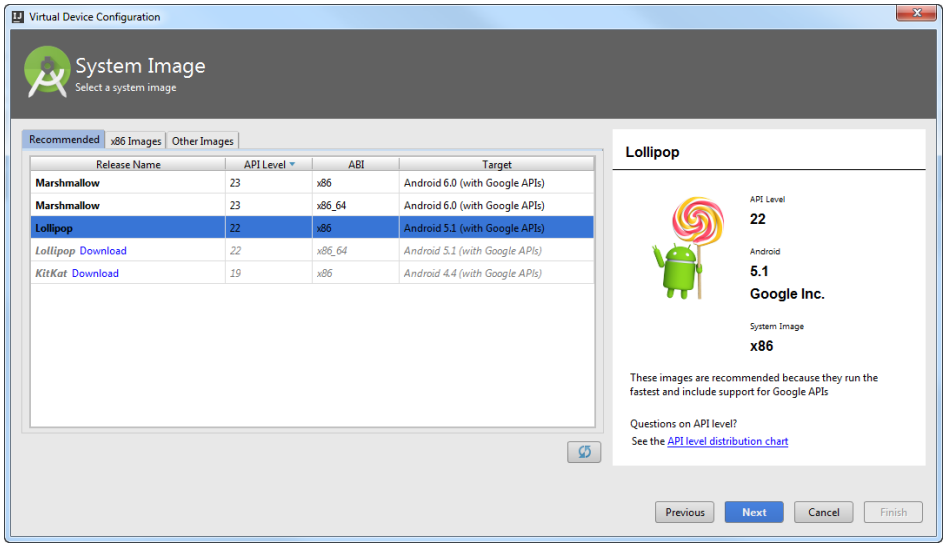
4. Next, make sure the Default APK option is selected under Installation Options . With this option selected, the application will be automatically deployed on the target device.
5. Next, you need to decide which activity you want to launch as a starter activity for your module. You can launch the default activity configured in the `AndroidManifest` file, or select a custom activity. Since your sample application only has one activity, select the Default Activity option.
6. Finally, you need to specify the target device where your application will be launched. Let's select the Emulator option. If you already have Android virtual devices configured, you can select a device from the Prefer Android Virtual Device drop-down list. If you have no devices configured, perform the following steps:
   1. Click the … button to launch the Android Virtual Device Manager:



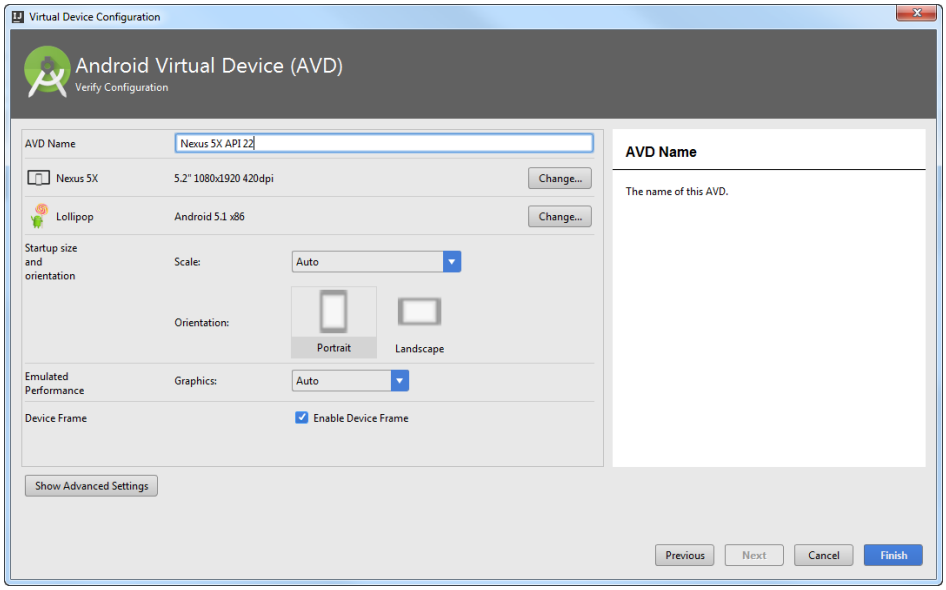
2. Click Create a virtual device. In the dialog that opens, in the left pane select the type of the Android device you want to mimic: TV, Phone, Wear or Tablet. Let's choose Phone. In the central pane select the phone model (for example, Nexus 5) and click Next:



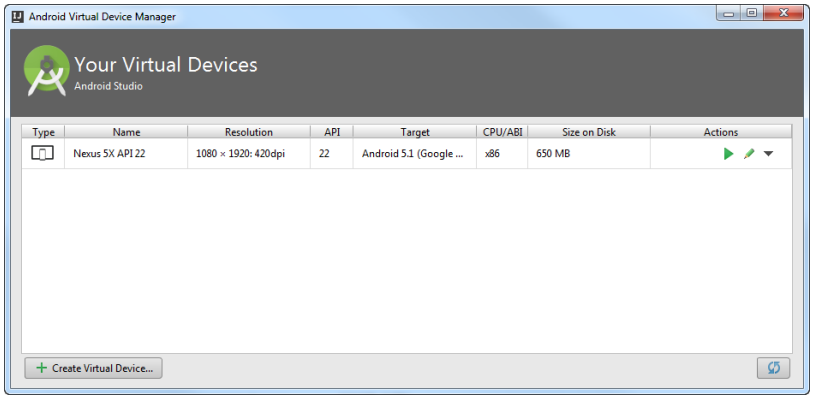
3. On the next step, select the system image that you want to mimic on the virtual device, i.e. the OS version, the Android API level, the application binary interface (ABI) and the target SDK version:



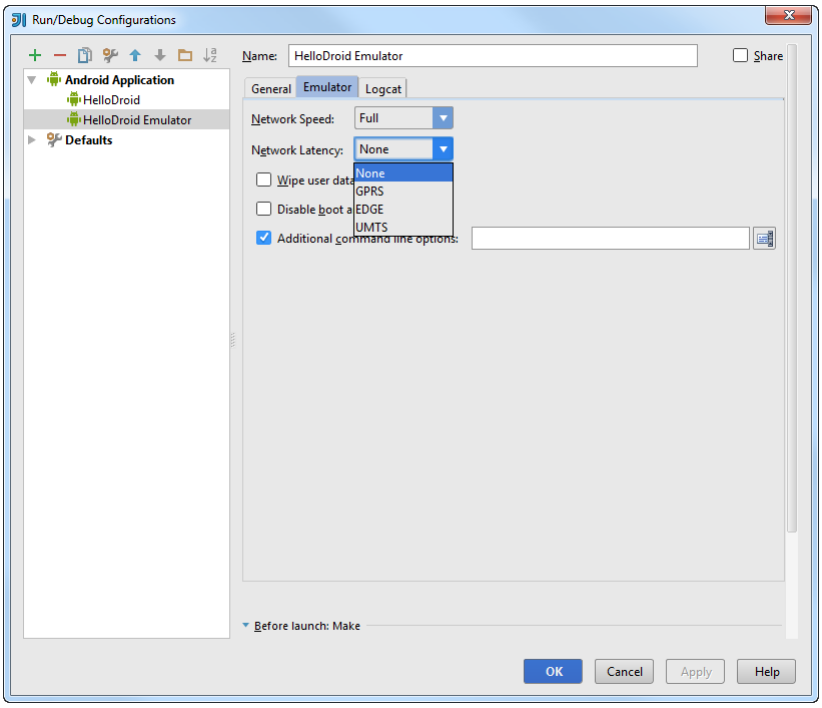
4. On the last step, you can modify the AVD name and select the startup size and orientation of the virtual device screen:



5. Click Finish to complete the wizard. The newly created Android virtual device will be added to the devices list:



6. Click OK to close the AVD Manager and return to the Run/Debug Configurations dialog.
7. Note that an Android Virtual Device (AVD) is just a configuration file through which you define the hardware and software options for the emulator to mimic. In addition to AVD settings, you can configure supplementary parameters through IntelliJ IDEA. Switch to the Emulator tab. You can set network transfer rate to be emulated, network latency (the time delay between the initial input and the output), etc. For a detailed description of the configuration options available in the Emulator tab, refer to Run/Debug Configuration: Android Application. Emulator tab .



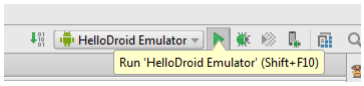
The parameters you can configure in the Emulator tab of the Run/Debug Configurations Dialog are then passed to the emulator as command line parameters. If needed, you can also specify additional command line parameters in the Additional command line options field.

8. In the Run/Debug Configurations dialog, click OK to save the newly created build configuration.
9. Note that if you selected an x86 application binary interface when configuring an Android virtual device, you will also need to install the Intel x86 Emulator Accelerator before you can run and test the application, otherwise IntelliJ IDEA will throw an error when you try to build your project. To do this:
   1. Launch the SDK Manager: in the main menu, select Tools | Android | SDK Manager .
   2. Scroll down to the folder called Extras.

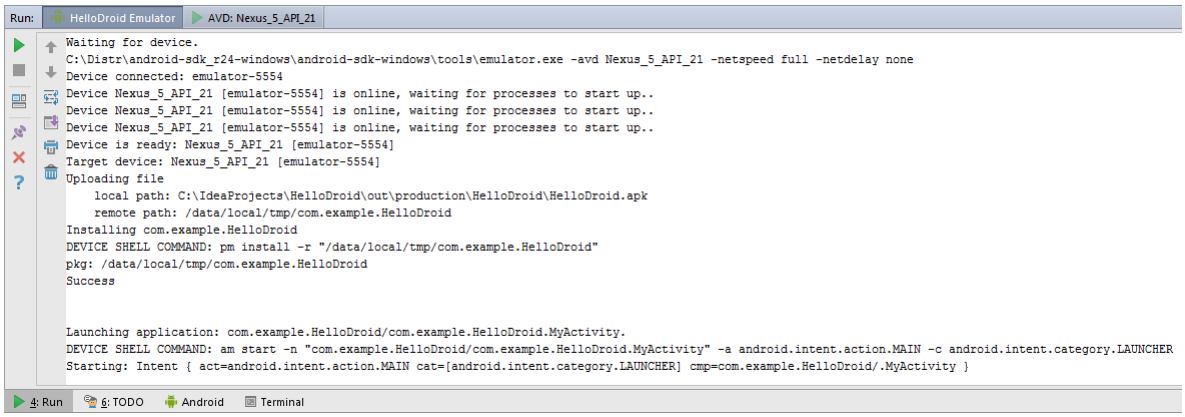
3. In this folder, locate Intel x86 Emulator Accelerator (HAXM installer), select the corresponding check box and click the Install packages button.
4. In the dialog that opens, accept the license agreement and click Install.
5. Navigate to the Android SDK installation directory (the SDK path is shown on top of the Android SDK Manager window). In the \extras\intel\Hardware_Accelerated_Execution_Manager folder, locate the `intelhaxm-android.exe` file and double-click to launch it.
6. Follow the instructions of the Intel Hardware Accelerated Execution Manager setup wizard to complete the installation.

Test the Application on an Emulator

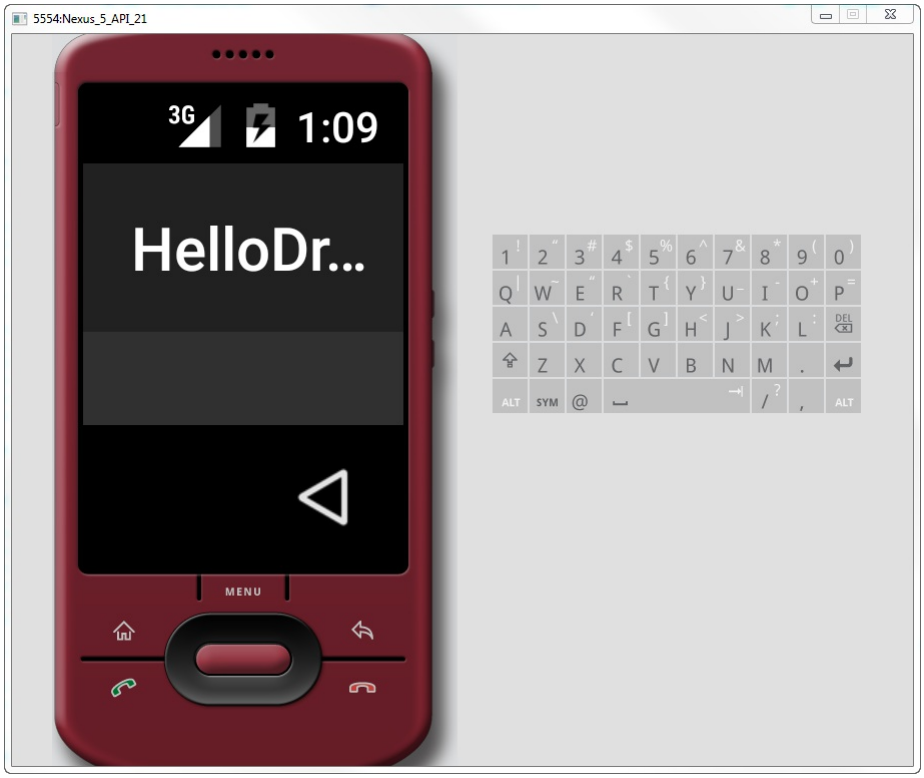
To build your project, press `Shift+F10` or click the Run icon in the Navigation bar in the top-right corner of the editor:



The project will be built according to the run/debug configuration currently selected in the drop-down list. If no errors occur, IntelliJ IDEA packages the binaries and resources into an APK file and uploads it to the Android emulator:



If the Android emulator is not already up and running, IntelliJ IDEA will initialize it before uploading the application. The emulator receives and installs the package and starts it by invoking the specified launch activity:



Test the Application on a Physical Device

To test your application on a real Android device, you need to:

1. Add a run/debug configuration and make sure that you select USB device under Target Device.
2. Make sure that an Android device is connected to the computer through a USB cable.
3. Press `Shift+F10` or click the Run icon in the Navigation bar in the top-right corner of the editor.

If the installation of your application fails, most probably this means that the device is not configured to install applications outside of the Android application stores. Enable this capability on your device to test the application.

On Android 4.2 or higher, do the following:

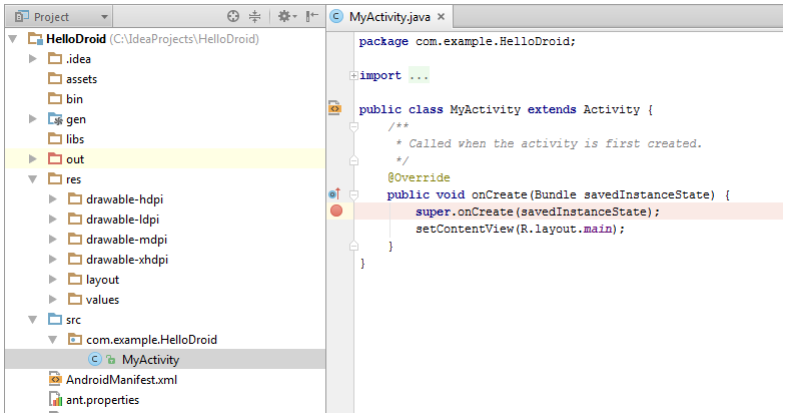
- Open your device's Settings.
- Scroll to About phone or About tablet and tap it.

- Scroll to the bottom and tap Build number 7 times until you see the "You are a developer!" message. By doing so, you've unlocked the USB debugging mode on your device.
- Now navigate to Settings | Developer Options | Debugging | USB Debugging to let your phone deploy non-packaged applications.

Debug your Application

IntelliJ IDEA allows you to launch your application in the debug mode on an emulator or a physical device using the corresponding run/debug configuration.

To debug your application, you need to set breakpoints in the source code: place the caret on an executable line and click the left gutter area. A red circle will appear next to the line where you want to toggle a breakpoint, and the line will be marked with pink:

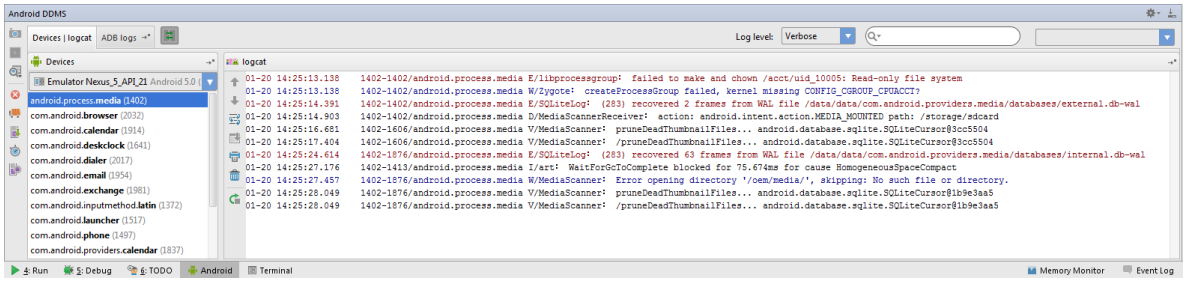


To start a debug session, press `Shift+F9` or click the Debug icon in the Navigation bar in the top-right corner of the editor. The Debug tool window will be activated where you can step through the program , examine variables , watches , frames and threads and analyse system information and error messages in the Console tab .

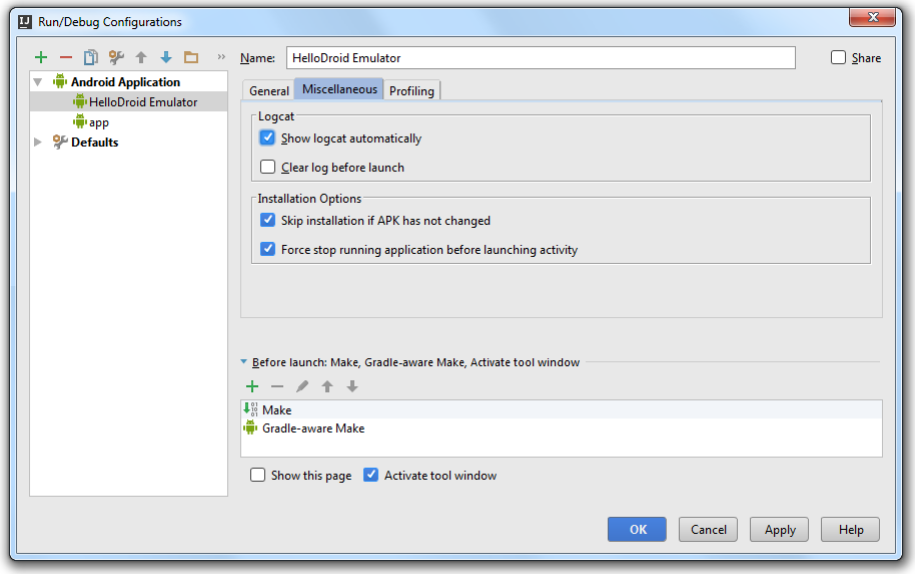
Analyze Debug Output

In IntelliJ IDEA, debugging of Android applications is provided through the support of the logcat functionality that stores a log of system debug output. Log messages include a stack trace when the emulator throws an error, so you can navigate to the exception location in the source code.

In IntelliJ IDEA, the logcat functionality is available through the Logcat tab of the Android tool window , or through the Logcat tab of the Debug tool window , the only difference being is that in the Debug tool window you can only view a log of all processes currently running on the connected device, while in the Android tool window you can filter log messages by the device and by a specific process running on the selected device (for details, see Debugging with Logcat):



You can also configure additional logcat options in the Miscellaneous tab of your run/debug configuration:



Editing UI Layout Using Designer

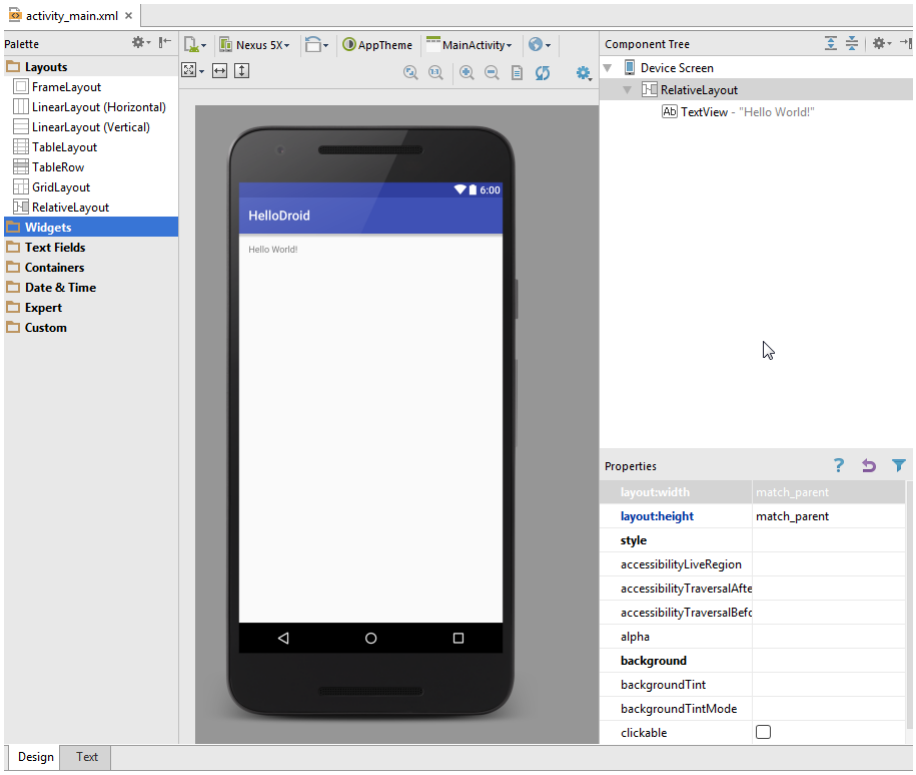
At this stage, the user interface of your sample `HelloDroid` application is based on a very simple layout. The UI layout is defined in the `activity_main.xml` file located in the `res/layout` folder. By default, IntelliJ IDEA provides a graphical view of a layout file, but it also lets you switch to a text-based view where you edit the layout file manually.

Let us modify the auto-generated user interface with the built-in UI Designer and see how the application layout is rendered without running it on any physical or virtual devices.

1. Open the layout file

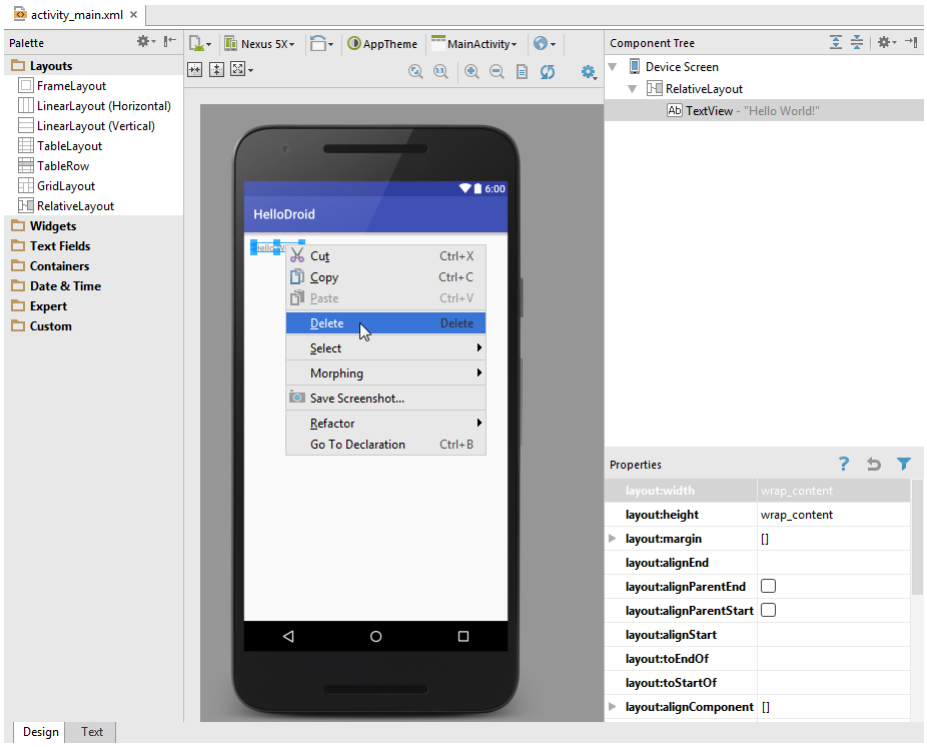
In the Project view , navigate to the `res\layout` folder and select the `activity_main.xml` file. The Design pane will open in the editor.

The pane shows a rectangular canvas that is synchronized with the current layout definition file and with the Component Tree view, so any changes to the canvas are reflected there accordingly.



2. Delete the existing text element

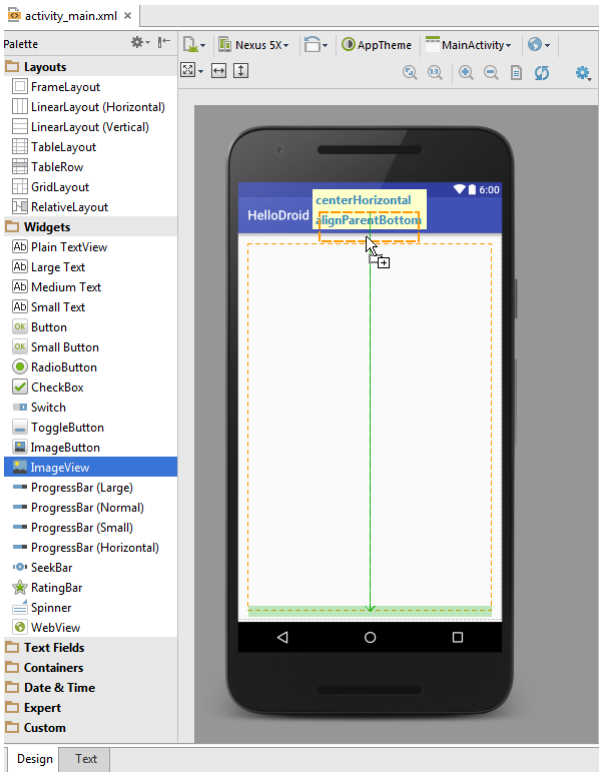
Let's delete the existing text element. To do this, click to select the text label in the view and right-click it to invoke the context menu. Select Delete to clear up the user interface:



3. Add an ImageView widget

Now add an ImageView widget: select the ImageView component from the Widgets palette and then click the canvas where you want to insert the widget. Alternatively, you

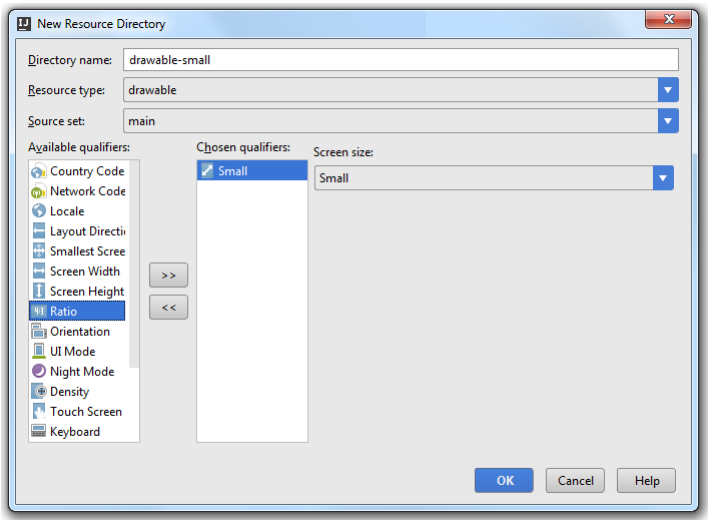
can drag-and-drop the widget to the Design pane:



At this point, the view contains a placeholder for an image, but there's no image associated with it yet. To add an image to the project, you first need to create a `drawable` folder under `res` .

4. Create the 'drawable' folder

Right-click the `res` node in the Project view and select New | Android resource directory. From the Resource type drop-down list, select drawable. If necessary, select any of the available qualifiers.



You use qualifiers such as 'small' only if you wish to use a different set of resources for different screens, UI modes, density or locales. If you are going to use a single set of resources, you need no qualifiers.

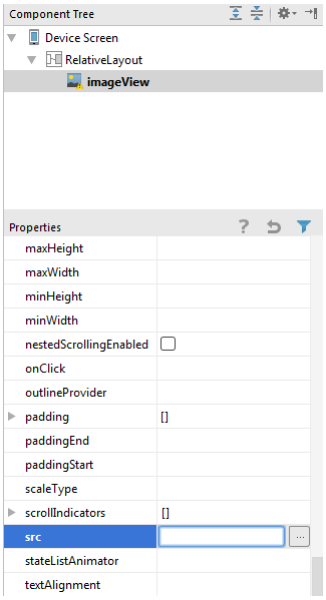
To add images, just pick image files in Windows Explorer and drag them to the `drawable` folder within IntelliJ IDEA.

5. Link an image file to the 'ImageView' widget

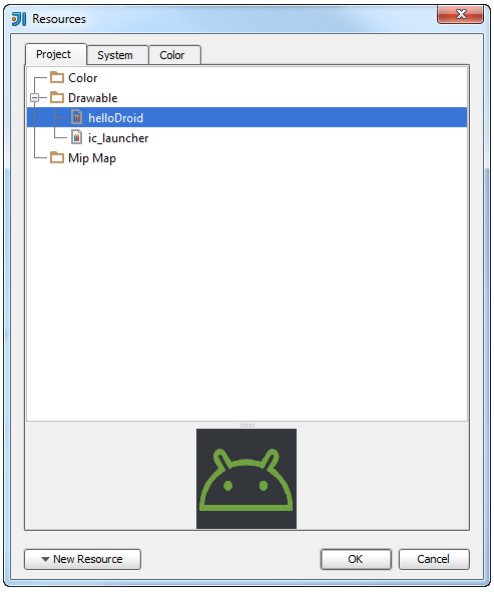
Now you need to link the image file you've added to the ImageView widget. In the UI designer, you need to select the widget to edit its properties. You can select the widget by clicking it, however, this sometimes can be a surprisingly hard task. If you have added an image widget but have no image attached to it, the widget is rendered as a very thin box that can be hard to select with the mouse. This is when Component Tree comes to a rescue and lets you easily select the visual elements you need.

Tip When it comes to adding, editing or removing graphical components of your UI layout, you can use the Component Tree in the same way as the Designer. You can drag-and-drop widgets onto the Component Tree and remove or edit elements from within the displayed hierarchy.

1. In the Component Tree pane, select the imageView component and locate the `src` entry in the table of its properties:



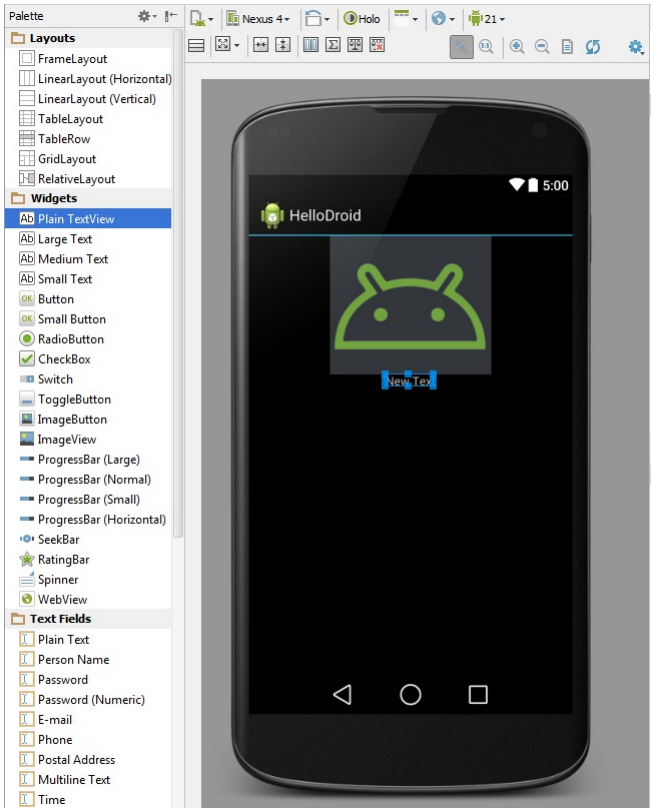
2. Click the Browse button ... and select the image you want to attach to the widget in the dialog that opens:



The image will be attached to the widget. You can select it in the designer and adjust its size and position:

6. Add a 'TextView' component

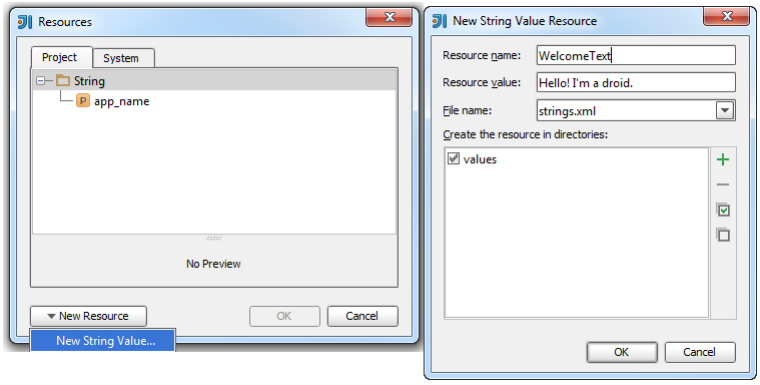
Now let us add a TextView component. In the palette, locate the Plain TextView component under Widgets and drag-and-drop it to the view just below the image:



By default, the Plain TextView displays some literal text: `New Text`. To change it and link it to some localizable string, you need to create a new text resource.

7. Create a 'String' resource

1. In the Components Tree, select the TextView element and locate the text property in the properties table below.
2. Click the Browse button [...] to invoke the Resources dialog box that allows you to pick an existing string value or add a new one.
   Note that strings are stored in the `strings.xml` file under the `res | values` folder. If necessary, you can edit the `strings.xml` file directly.
3. Click New Resource | New String Value. In the dialog that opens, enter the resource name and specify the text that will be displayed in the TextView widget:

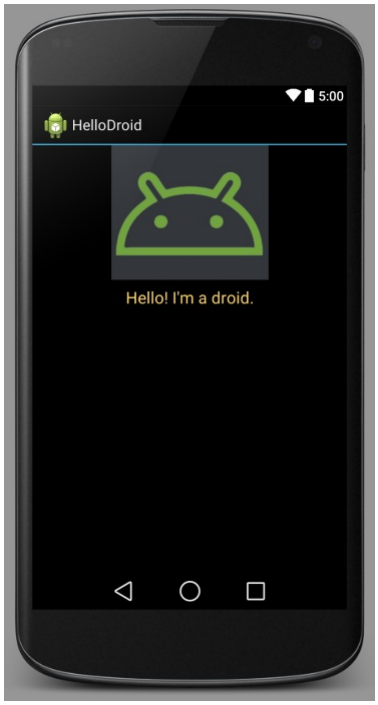


8. Add style to the text

To make the text look a bit more appealing, you need to set a few additional properties. You can do this by editing properties of the TextView component. Let's do the following:

1. Center the label horizontally: set the gravity property to `center_horizontal`.
2. Pad the text a bit: locate the padding property and set the value of all to `10dp`.
3. Change the font color: edit the textColor property. You can set the property value to a color string, such as `#ffd764`, for example, or you can have it reference a color resource. To add a color resource, click the Browse button [...] and create a resource named `welcomeText` with the value of `#ffd764`.
4. Change the font size: edit the TextSize property. You can set a value, or link it to a size resource in the same way we did with the TextColor property above.
   Note that you cannot indicate dimensions as plain numbers. You must always add the dp suffix.

As a result, your user interface now looks as following:



9. Preview your layout in various conditions

The controls on top of the Designer tool window allow you to preview your UI layout in different conditions: landscape or portrait, on different screen sizes, using different themes, for different locales, etc. This provides a quick and easy way for you to see how the overall application UI looks like in a number of common scenarios.

For example, do the following:

- Click the icon and select Preview All Screen Sizes from the drop-down menu. IntelliJ IDEA will display the preview of your UI layout on the most common screen types.
- To preview the application layout on different devices, click the icon with the current device name (in our case, it is Nexus 5) and choose from the list.
- Click to toggle between the layout and portrait preview mode.

Editing UI Layout Using Text Editor

Now when you are familiar with the basic UI layout editing options using the built-in designer , let's add more elements to the main view of the application by editing its layout manually.

1. Switch to Text view

Switch to the Text tab at the bottom of the editor. IntelliJ IDEA will display the XML source code of the currently selected layout file:

2. Add a horizontal ruler

Let's add some markup that inserts a separator. The easiest way to add a horizontal dividing line is adding the following to your source code:

```
<View android:layout_width="fill_parent"
                    android:layout_height="5dp"
                    android:layout_marginTop="60dp"
                    android:background="#00ff00" />
```

The separator is 5 units thick, painted with a green background and placed below the nearest element.

3. Add a TextView element

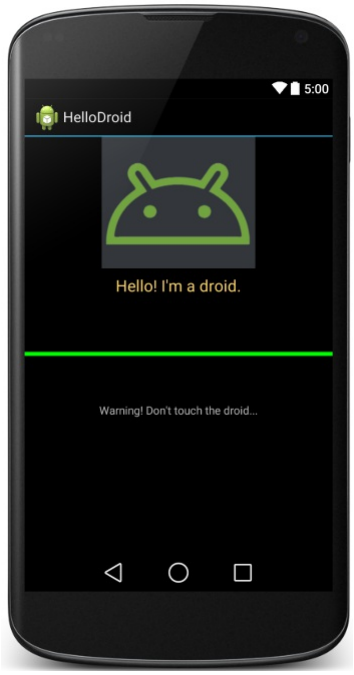
To add another TextView element, add the following to your source code:

```
<TextView android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="60dp"
                    android:text="Warning! Don't touch the droid..."
                    android:id="@+id/message"
                    android:layout_gravity="center_horizontal" />
```

The new element will be placed 60 units below the separator; it is centered horizontally and uses the default font color. The text string associated with the element is declared explicitly.

Unlike the separator, though, the TextView element also has the id property. Its syntax indicates that what follows the "/" symbol, is a string that must be treated as an ID resource, and will be used to reference the view element. The Android runtime processes this information appropriately and makes it possible for you to write Java code that interacts with the TextView component.

The Preview pane on the right displays the result of your changes:

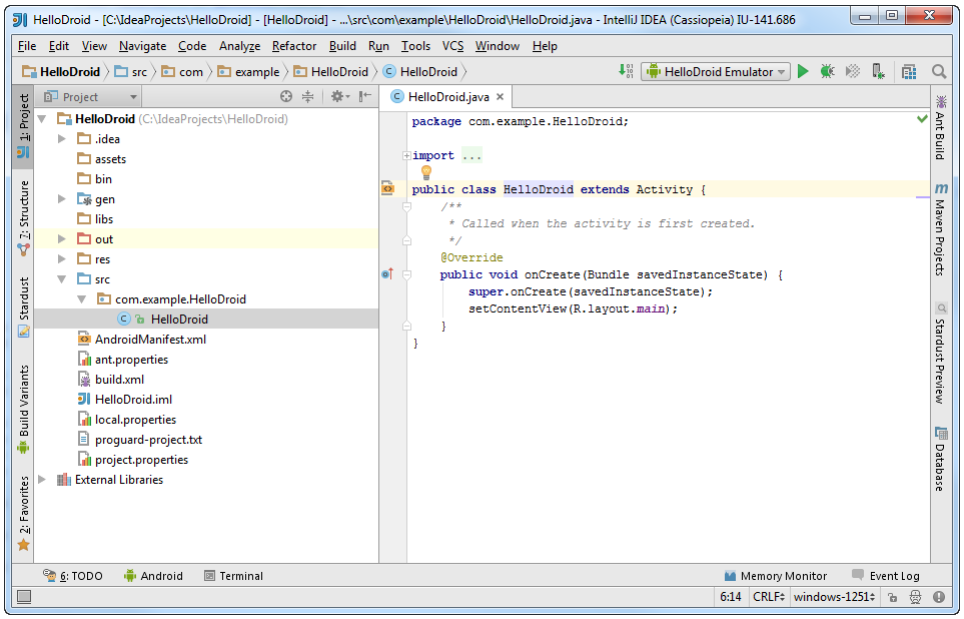

5:00
HelloDroid
Hello! I'm a droid.
Warning! Don't touch the droid...

Making the Application Interactive

Although our sample application is fully functional at this point, it does not support any form of interaction yet. To make the application support tap events, you need to edit the HelloDroid class defined in the `src` folder.

1. Open the MyActivity class

In the Project view, locate the `HelloDroid.java` file under the `src | com.example.HelloDroid` folder and click on it:



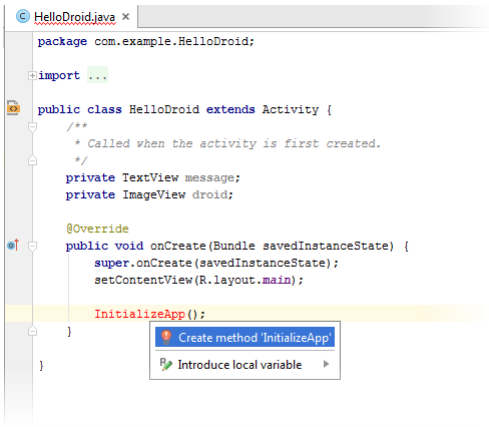
2. Add references to visual elements

In Android, you need to explicitly obtain references to visual elements in order to manipulate them programmatically. You need to define private members on the activity class to hold these references, and then initialize these members in a newly created method invoked from within `onCreate`.

1. Add the following code to the `HelloDroid` class:

```
private TextView message;
                    private ImageView droid;
```

2. Add a call to a new method called `InitializeApp` in `OnCreate`. IntelliJ IDEA promptly detects that this method is missing and suggests generating it for you:



3. In the `InitializeApp` method, assign private members a reference to a visual element:

```
message = (TextView)
                    findViewById(R.id.message);
                    droid = (ImageView)
                    findViewById(R.id.imageView);
```

The expression `R.id.xxx` indicates a member of the auto-generated R class.

3. Add an event handler

In an application, no interaction is possible without events and event handlers. As an example, let's add a click handler to the droid image view and display a message every time the user touches the image.

In Java, an event handler takes the following form:

```
private View.OnClickListener
                    droidTapListener;
```

Add this member to the `HelloDroid` class and initialize it in the `InitializeApp` method. You code should now look as following:

```
private void InitializeApp()
                    {
                    message = (TextView)
                    findViewById(R.id.message);
                    droid = (ImageView)
                    findViewById(R.id.imageView);

                    // Define and attach listeners
                    droidTapListener = new View.OnClickListener()  {
                    public void onClick(View
                    v) {
                    TapDroid();
                    }
                    };
                    droid.setOnClickListener(droidTapListener);
                    }
```

The net effect of this code is that every time the user taps the image, the `TapDroid` method is invoked.

4. Handle the 'Click' event

1. The `TapDroid` method just counts the times the user touched the image, and displays a message. You need to add a new private member to the `HelloDroid` class to count clicks:

```
public class MyActivity
                    extends
                    Activity
                    {
                    private TextView message;
                    private ImageView droid;
                    private View.OnClickListener
                    droidTapListener;
                    private int counter = 0;

                    // More code goes here ...
                    }
```

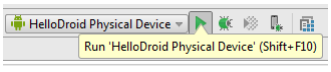
2. Next, define the `TapDroid` method as shown below:

```
private void TapDroid() {
                    counter++;
                    String temp;
                    switch (counter)
                    {
                    case 1:
                    temp = "once";
                    break;
                    case 2:
                    temp = "twice";
                    break;
                    default:
                    temp = String.format("%d times", counter);
                    }
                    message.setText(String.format("You touched the droid %s", temp));
                    }
```

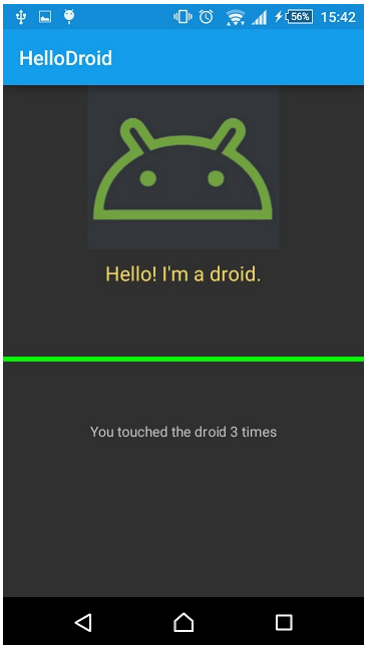
5. Build an application and launch it on a device

Your sample application is now complete. You can build and deploy it to an Android device.

1. Create a run/debug configuration and select USB device under Target Device.
2. Connect an Android device to the computer through a USB cable. If the device is connected for the first time, wait until all drivers are installed.
3. If this is the first time you are deploying an application outside of the Android application stores, enable the USB Debugging mode on your device.
   On Android 4.2 or higher, do the following:
   - Open your device's Settings.
   - Scroll to About phone or About tablet and tap it.
   - Scroll to the bottom and tap Build number 7 times until you see the "You are a developer!" message. By doing so, you've unlocked the USB debugging mode on your device.
   - Now navigate to Settings | Developer Options | Debugging | USB Debugging to let your phone deploy non-packaged applications.
4. Make sure the appropriate run/debug configuration is selected in the drop-down in the top-right corner of the editor and click the Run icon:



5. When the application has been successfully deployed on the device, tap the image and look at the changes to the user interface:


15:42
HelloDroid
Hello! I'm a droid.
You touched the droid 3 times

Creating Unit Tests

In Android, unit testing is based on JUnit , and plane use of JUnit is enough to test the features that are exclusively based on Java code.

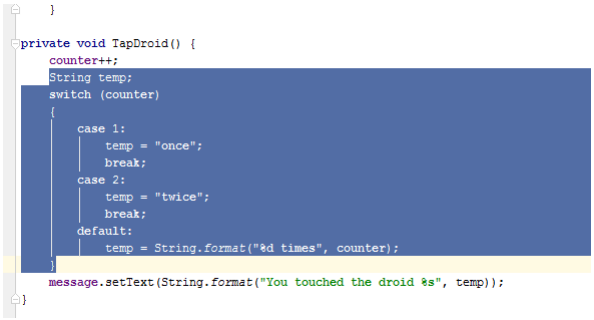
However, to test Android-specific functionality you need a bunch of wrapper classes built on top of JUnit. IntelliJ IDEA streamlines most of the tasks around the build of an Android test project.

1. Make sure your code is testable

Unit testing requires that the source code is composed in such a way that dependencies between modules can be easily neutralized with mocks. In addition, unit testing requires that functions are well isolated from each other.

As is, the code of the `HelloDroid` class is not easy to test. Let's first apply a quick refactoring before we proceed with unit tests.

1. Open the `HelloDroid` class and select the portion of the code in the `TapDroid` method that refers to the production of the display message:



```
    }

private void TapDroid() {
    counter++;
    String temp;
    switch (counter)
    {
        case 1:
            temp = "once";
            break;
        case 2:
            temp = "twice";
            break;
        default:
            temp = String.format("%d times", counter);
    }
    message.setText(String.format("You touched the droid %s", temp));
}
```

2. Rewrite the `TapDroid` method in such a way so that it calls into a newly created public helper method (`GetStringFor Display`) as shown below:

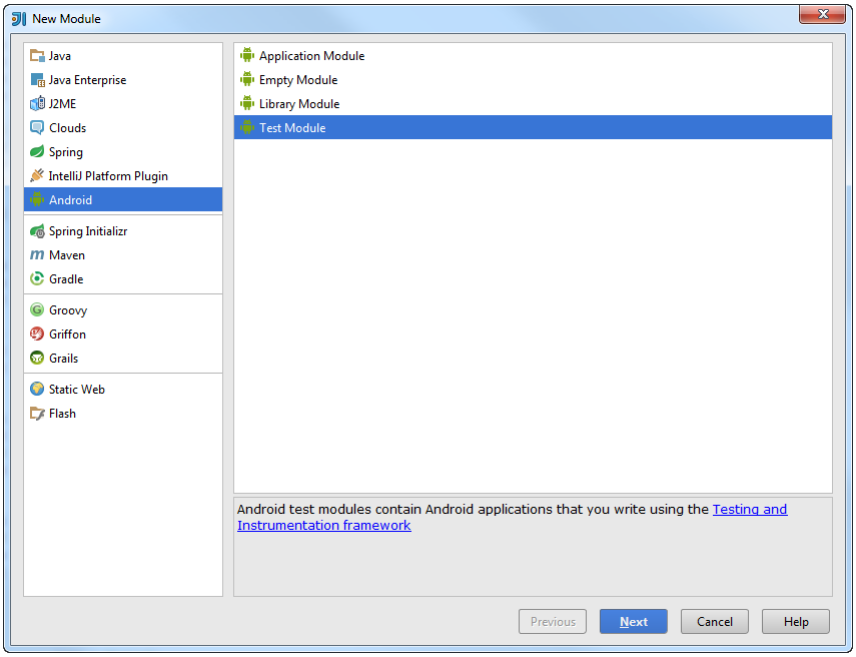
```
private void TapDroid() {
                    counter++;
                    String temp = getStringForDisplay(counter);
                    message.setText(String.format("You touched the droid %s", temp));
                    }

                    public String getStringForDisplay(int count) {
                    String temp;
                    switch(count)
                    {
                    case 1:
                    temp = "once";
                    break;
                    case 2:
                    temp = "twice";
                    break;
                    default:
                    temp = String.format("%d times", count);
                    }
                    return temp;
                    }
```

The `getStringForDisplay` method is now much easier to test, and the body of the `TapDroid` method has been greatly simplified.

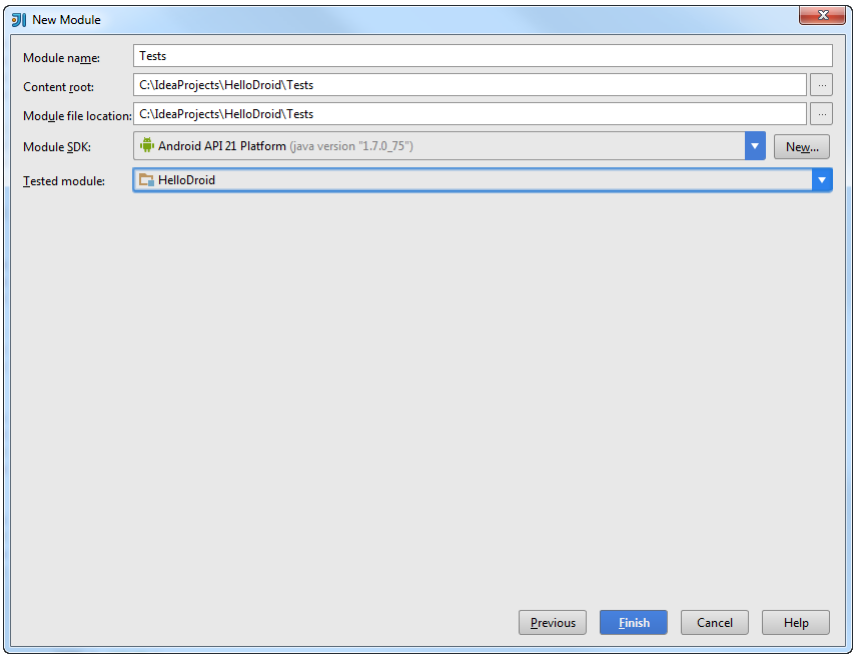
2. Create a test module

Now let's create a new test module and set `HelloDroid` as the tested module. This ensures that the test module holds a reference onto the module that contains the source code you are going to test.

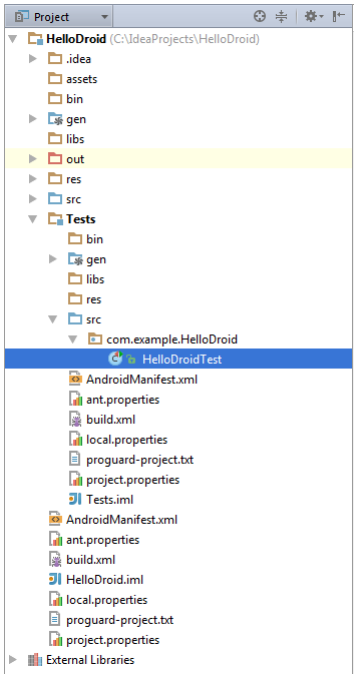
1. From the main menu, select File | New | Module to launch the New Module wizard .
2. On the first page of the wizard, select Android in the left pane, and Test Module on the right:



3. On the second page, specify the new module name, for example, `Tests`. Leave the other fields unchanged. The `HelloDroid` module is specified as the tested module automatically, as at this point, this is the only module in the project.



A new node will be appended to the project named Tests. This module has its own `manifest` file and and src directory. The `manifest` file links against the `android.test` library in order to build test classes.



The newly created module has a test file named `HelloDroidTest` in the src folder. You can add more test files simply by adding more Java classes as shown below:

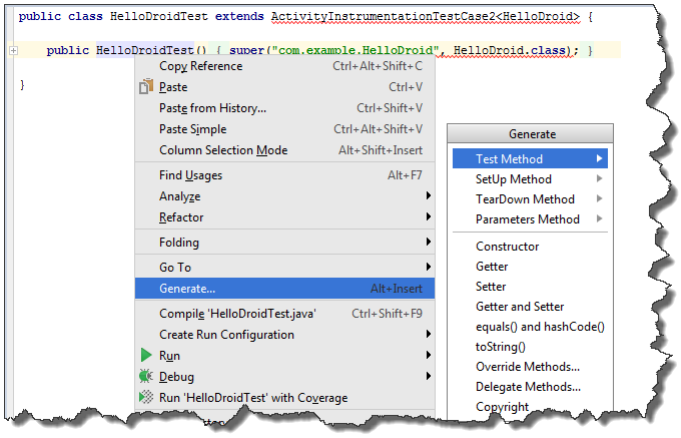
```
public class HelloDroidTest
                    extends
                    ActivityInstrumentationTestCase2<HelloDroid>
                    {

                    public HelloDroidTest() {
                    super("com.example.HelloDroid",
                    HelloDroid.class);
                    }
                    }
```

The test class inherits from `ActivityInstrumentationTestCase2<T>` where `T` is the name of the activity you are going to test.

Note that adding a constructor is required, as there is no default constructor defined for the parent class.

3. Add a test method

In the editor, right-click the `HelloDroid` test class and click Generate (alternatively, click `Alt+Insert`). From the popup menu that opens, select Test Method:



IntelliJ IDEA creates a new method stub named `testName` where you can easily change the `Name` suffix into something more meaningful in the context:

```
public void testStringForDisplay() throws Exception {
```

The `test` prefix in the method name is required if you are using JUnit 3 , the default testing framework in Android. With JUnit 4 , you have to use method name annotations to indicate that a given method must be processed as a test method.

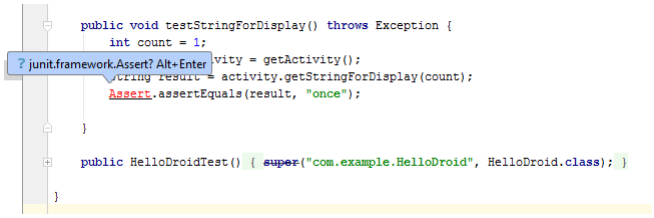
4. Write the logic for a test method

Internally, the test method first gets a reference to the activity it is trying to test, then it calls the target method, and, finally, compares the effective results with the expected results.

Add the following code to the test method:

```
public void testStringForDisplay()
                    throws Exception {
                    int count = 1;
                    HelloDroid activity = getActivity();
                    String result = activity.getStringForDisplay(count);
                    Assert.assertEquals(result, "once");

                    }
```

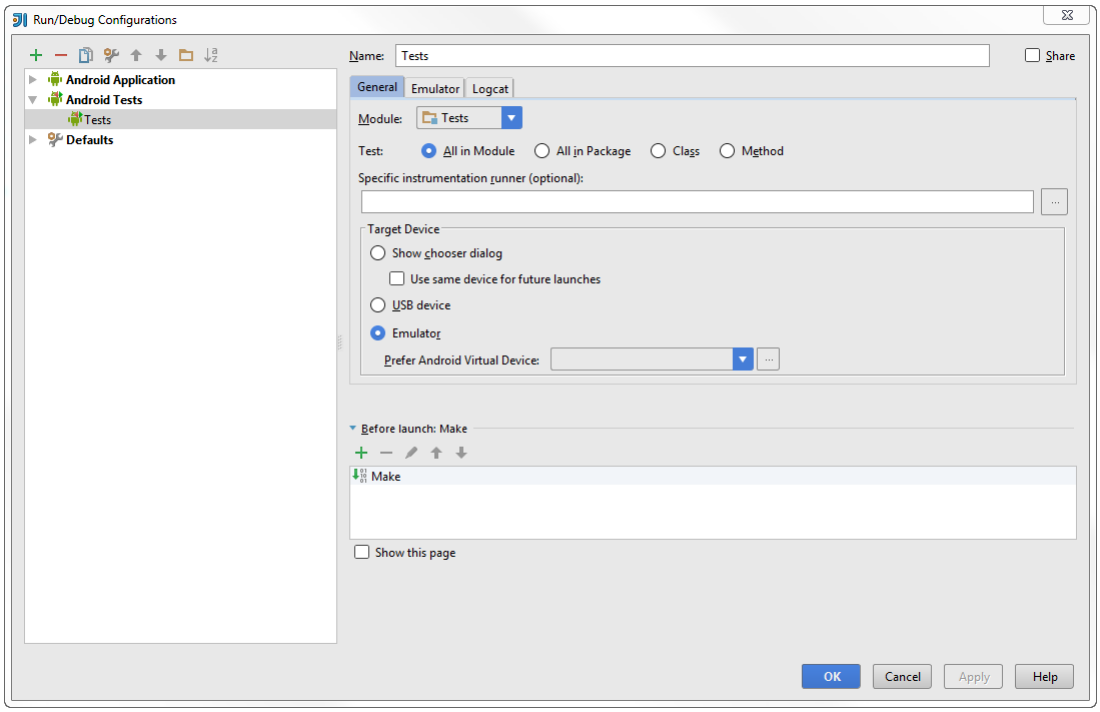
Assertions are implemented through the services of the JUnit framework and need to be properly referenced in the source file. Press `Alt+Enter` when the intention action pops up to reference it:



5. Create a run/debug configuration for tests

In order to run tests, you need to create a dedicated run/debug configuration. A default configuration is created for you automatically when you set up a test module.

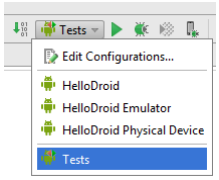
To edit its settings, in the main menu select Run | Edit Configurations and select Tests under Android Tests in the left pane:



You can select to run all tests in the module, or limit the test to the methods in a given class.

6. Run a test

To run your tests, make sure the appropriate run/debug configuration is selected in the drop-down list in the top-right corner of the editor, and click the Run button next to it:



Test results are displayed in the Test Runner tab of the Run tool window that is activated automatically. If a test is completed successfully, a green square icon appears in the top right corner of the editor. If there are warnings, the icon is yellow, and if a test fails - it is red. You can click the icon to get more details.

You can export a test report to a variety of formats by clicking the Export Test Results icon in the Tests tab toolbar .

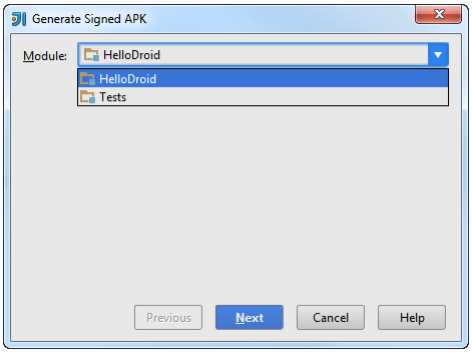
Packaging the Application

Once ready, an Android application is packaged as an .apk file. The package contains binaries as well as resources.

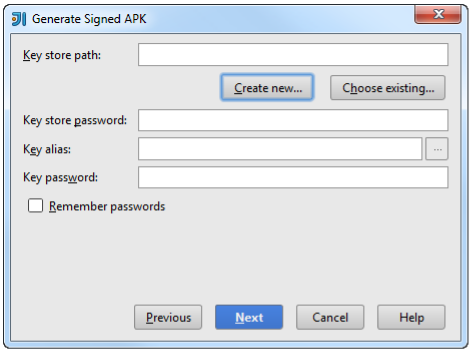
Android applications can be published in an application store, such as Google Play. To publish an application, you need to sign it digitally. Based on this signature, the Android system identifies the author of every deployed application. You do not need to apply for a personal signature to any authority, a signature generated by IntelliJ IDEA is quite sufficient.

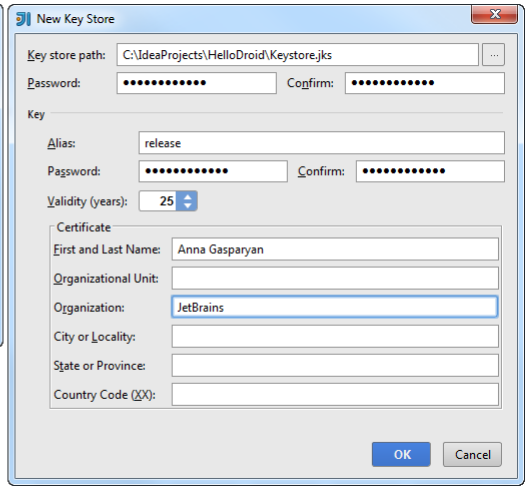
To package and sign the application, perform the following steps:

1. From the main menu, select Build | Generate Signed APK. The Generate Signed APK Wizard starts.
2. On the first page, select the module that you want to package (`HelloDroid`) and click Next:

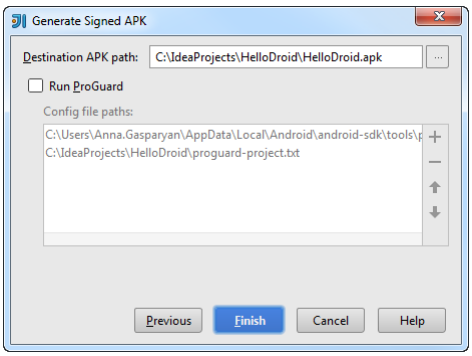


3. On the next page, click the Create new button to create a new keystore file with a new key.
   - In the New Key Store dialog box that opens, click the Browse button ..., select the folder where the newly generated keystore file will be located, and enter the name of the file.
   - Specify and confirm the password to access the keystore.
   - Configure the new release key by filling in the data in the Key area.





4. On the next step, in the Destination APK path text box, specify the folder where the `.apk` file will be saved. Optionally, select the Run ProGuard option if you want IntelliJ IDEA to obfuscate the application during the packaging procedure.



5. Click Finish to generate and sign the application package.

You can upload the resulting `HelloDroid.apk` file to an application store, or install it directly on devices configured to install applications from unknown sources.

Arquillian: a Quick Start Guide

This guide shows main IntelliJ IDEA features for writing and running Arquillian tests.

- Before you start
- Creating a project with Arquillian JUnit support
- Creating a class
- Developing code for the Greeter class
- Defining javax-inject.jar as a library
- Creating a folder for test sources
- Creating a test class
- Completing the code for the GreteerTest class
- Creating a run configuration for running the test
- Running the test in an embedded container
- Editing the run configuration: adding a managed container
- Creating the arquillian.xml configuration file
- Running the test in a managed container
- Modifying arquillian.xml
- Running the test: deploying to a running server

Before you start

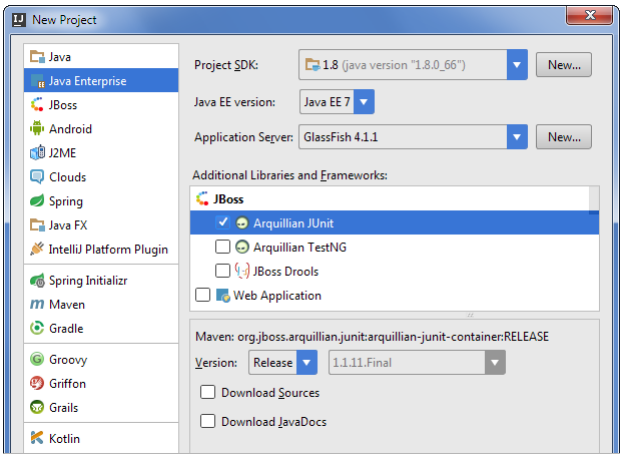
Make sure that the following software is installed on your computer:

- IntelliJ IDEA ULTIMATE Edition. Also check if the JBoss Arquillian Support plugin is enabled.
- Java SE Development Kit (JDK), version 8. Download Oracle JDK.
- GlassFish Server, version 4. The server will be used as a managed Arquillian container. Download GlassFish.

You should also download javax-inject.jar. This file will be used as a library when developing our sample test class.

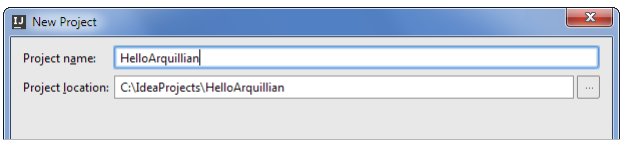
Creating a project with Arquillian JUnit support

1. Click Create New Project on the Welcome screen, or select File | New | Project.
   The New Project wizard opens.
2. In the left-hand pane, select Java Enterprise.
3. If the JDK that you want to use is already defined in IntelliJ IDEA, select that JDK from the Project SDK list. Otherwise, click New, select JDK, and select the JDK installation folder in the dialog that opens.
4. If GlassFish is not defined in IntelliJ IDEA yet, click New to the right of the Application Server field and select Glassfish Server.
   In the Glassfish Server dialog, specify the GlassFish Server installation directory.
5. Under Additional Libraries and Frameworks, select the Arquillian JUnit check box.

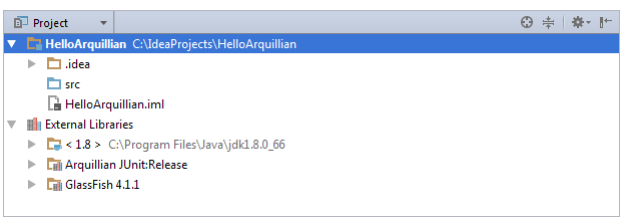


Click Next.

6. Specify the name for your new project (e.g. `HelloArquillian`) and click Finish.



When the project is created, you'll see something similar to this in the Project tool window.

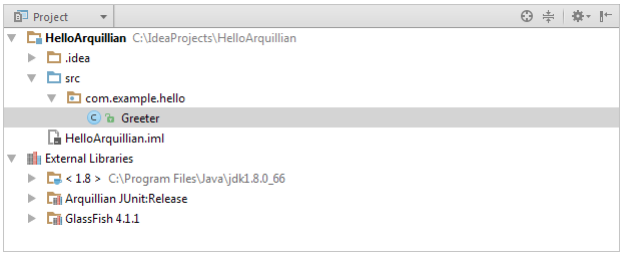


(To add Arquillian JUnit support for an existing project: in the Project tool window, right-click your project or module folder and select Add Framework Support. Then, select the Arquillian JUnit check box in the dialog that opens.)

Creating a class

Now we are going to create a class that we'll test. Let the class name be `com.example.hello.Greeter`.

1. In the Project tool window, right-click the `src` folder, point to New and select Java Class.
2. In the Create New Class dialog that opens, type `com.example.hello.Greeter` in the Name field and press `Enter`.
   The package `com.example.hello` and the class `Greeter` are shown in the Project tool window.



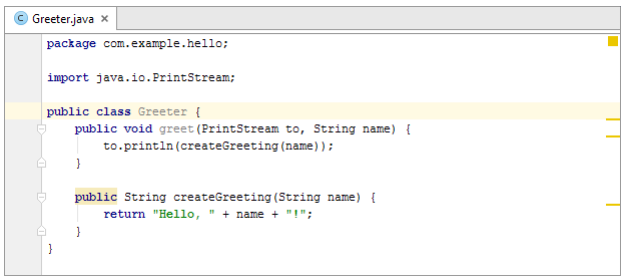
At the same time, the file `Greeter.java` opens in the editor.

Developing code for the Greeter class

Here is the code for the `Greeter` class.

```
package com.example.hello;

                    import java.io.PrintStream;

                    public class Greeter {
                    public void greet(PrintStream
                    to, String name)
                    {
                    to.println(createGreeting(name));
                    }

                    public String createGreeting(String name) {
                    return "Hello, " + name
                    + "!";
                    }
                    }
```

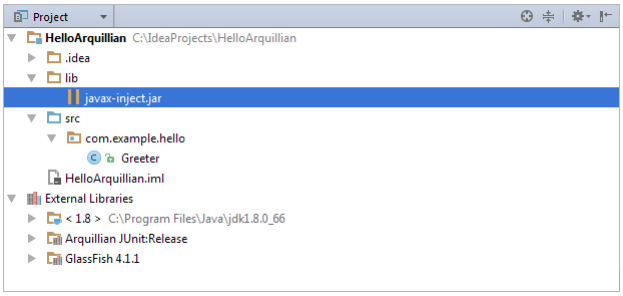
Copy the code into the editor.



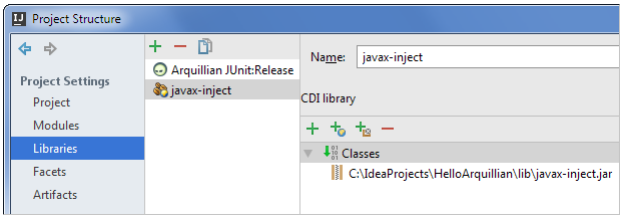
Defining javax-inject.jar as a library

To be able to write and run our Arquillian test, we need `javax-inject.jar` as a library.

1. In the project root folder, create the folder `lib` (New | Directory) and copy `javax-inject.jar` into that folder.



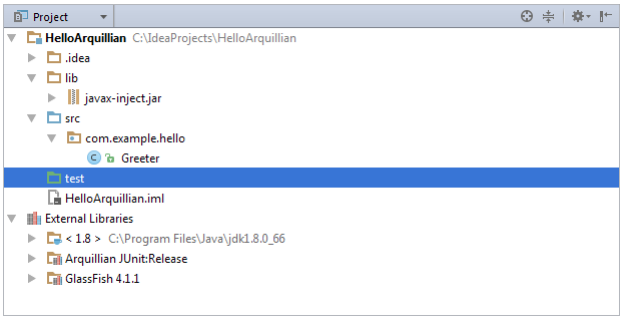
2. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`) and select Libraries.
3. Click , select Java and select `javax-inject.jar` in the dialog that opens.
4. Click OK in the Choose Modules dialog.



5. Click OK in the Project Structure dialog.

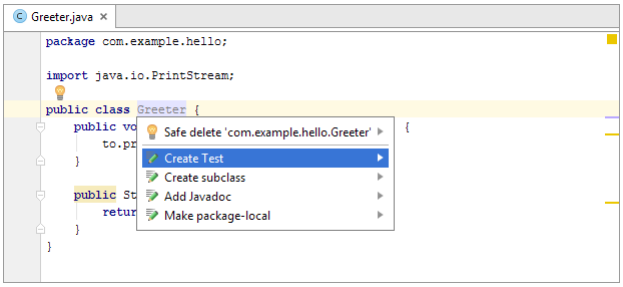
Creating a folder for test sources

1. In the project root folder, create the folder `test`.
2. Right-click that folder, point to Mark Directory As and select Test Sources Root.

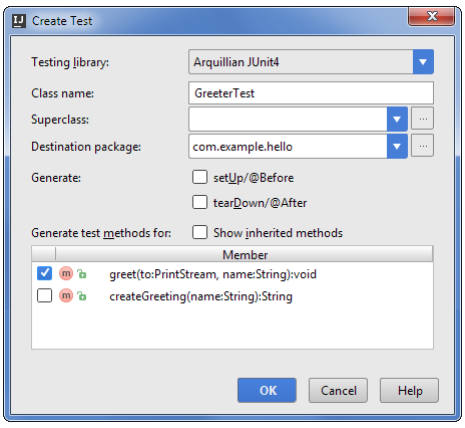


Creating a test class

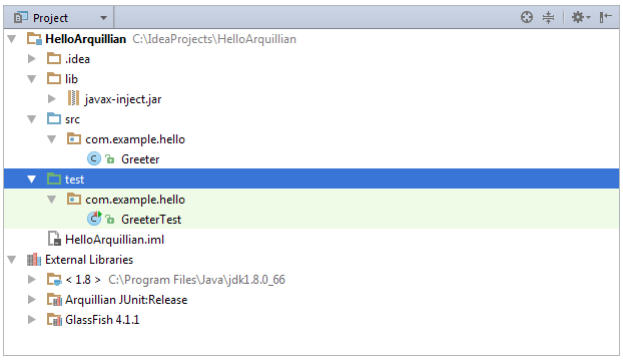
1. In the editor, place the cursor within the name of the class (`Greeter`).
2. Click the light bulb (`Alt+Enter`) and select Create Test.



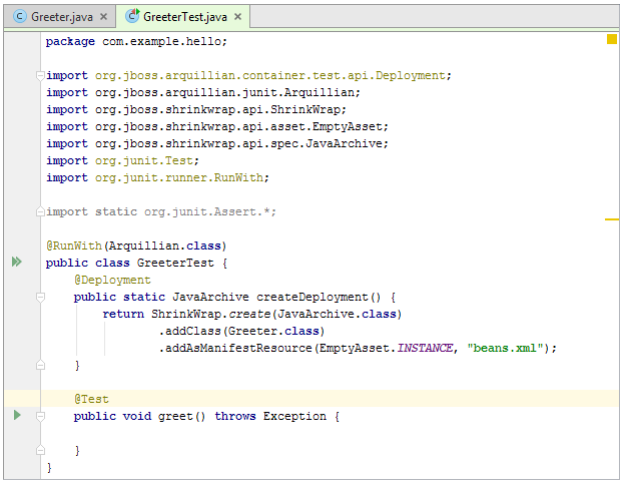
3. In the Create Test dialog that opens, select Arquillian JUnit4 from the Testing library list. Under Create test methods for, select the method that doesn't return a value (`greet`). Click OK.



The new test class is shown in the Project tool window.



At the same time, the file `GreeterTest.java` opens in the editor.



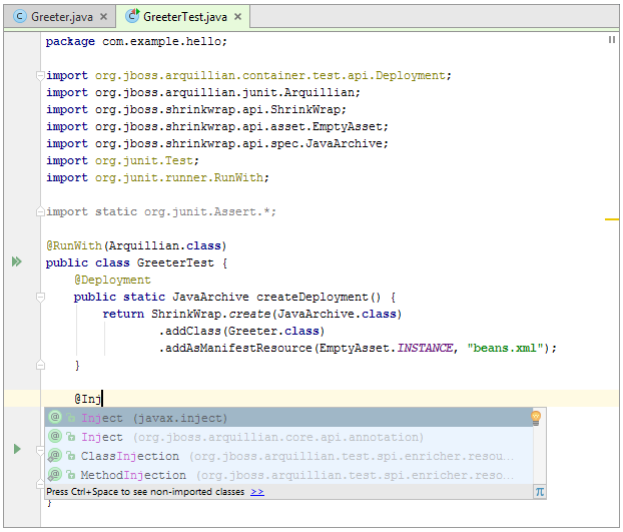
The initial content of the test class is defined by the corresponding template. You can edit that template on the Code tab of the File and Code Templates page in the Settings / Preferences dialog (`Ctrl+Alt+S` | Editor | File and Code Templates).

Completing the code for the GreteerTest class

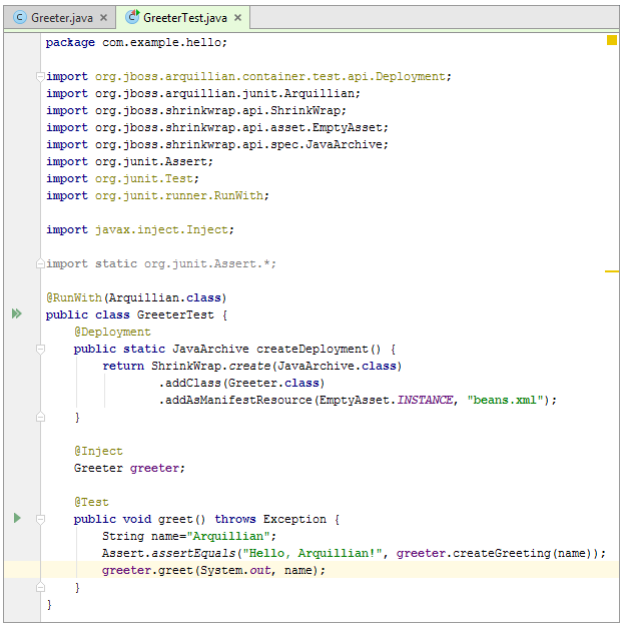
Here is the code for the test class in its final state:

```
package com.example.hello;

                    import org.jboss.arquillian.container.test.api.Deployment;
                    import org.jboss.arquillian.junit.Arquillian;
                    import org.jboss.shrinkwrap.api.ShrinkWrap;
                    import org.jboss.shrinkwrap.api.asset.EmptyAsset;
                    import org.jboss.shrinkwrap.api.spec.JavaArchive;
                    import org.junit.Assert;
                    import org.junit.Test;
                    import org.junit.runner.RunWith;

                    import javax.inject.Inject;

                    import static org.junit.Assert.*;

                    @RunWith(Arquillian.class)
                    public class GreeterTest {
                    @Deployment
                    public static JavaArchive createDeployment() {
                    return ShrinkWrap.create(JavaArchive.class)
                    .addClass(Greeter.class)
                    .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
                    }

                    @Inject
                    Greeter greeter;

                    @Test
                    public void greet() throws
                    Exception {
                    String name="Arquillian";
                    Assert.assertEquals("Hello,
                    Arquillian!", greeter.createGreeting(name));
                    greeter.greet(System.out, name);
                    }
                    }
```

1. To insert a proper `import` statement for `@Inject`, type `@Inj` and select @Inject (javax.inject).

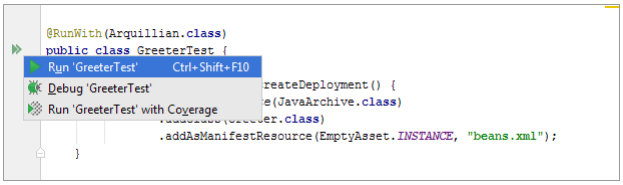


2. Add the remaining code by copying.

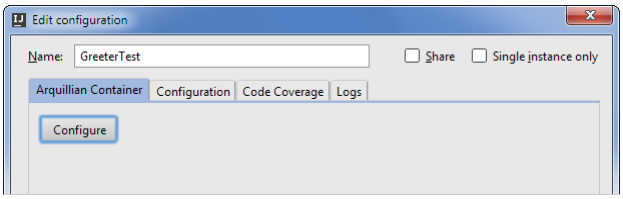


Creating a run configuration for running the test

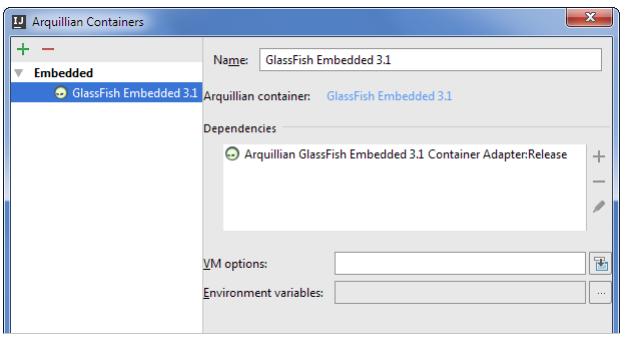
1. To the left of `public class GreeterTest`, click ▶▶ and select Run 'GreeterTest'.



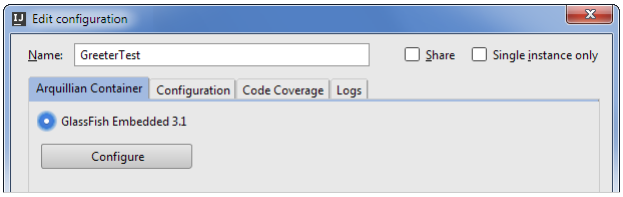
2. In the Edit configuration dialog that opens, click Configure.



3. In the Arquillian Containers dialog, click +, point to Embedded and select GlassFish Embedded 3.1. (We'll start by running the test in an embedded container.)

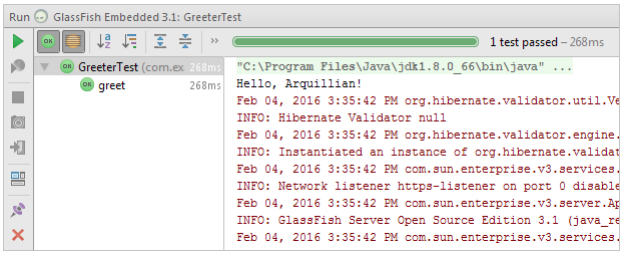


4. In the Edit configuration dialog, select GlassFish Embedded 3.1.



Running the test in an embedded container

1. In the Edit configuration dialog, click Run.
   The Run tool window opens and, after some time, the test result is shown there.

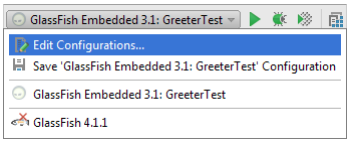


2. Close the Run tool window by clicking ✖.

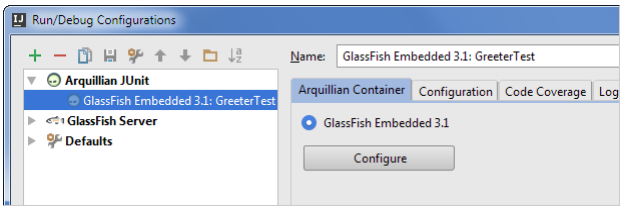
Editing the run configuration: adding a managed container

Now let's change our run configuration so that it could be used for running the test in a managed container.

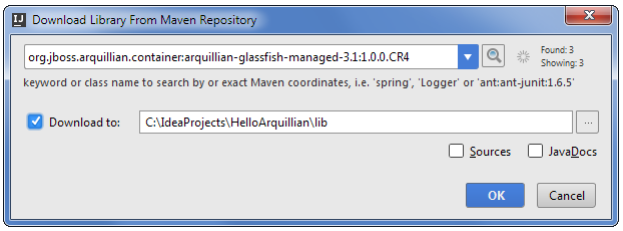
1. Click the run configuration selector and select Edit Configurations.



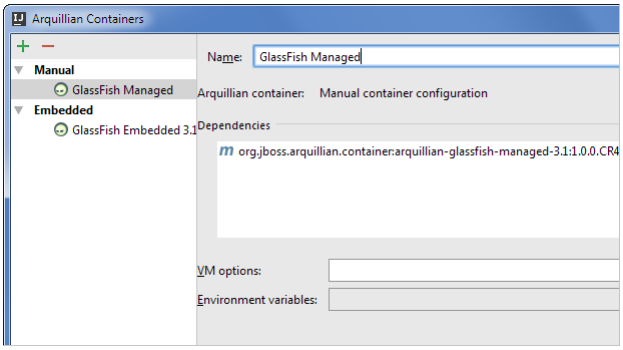
2. In the Run/Debug Configurations dialog, click Configure.



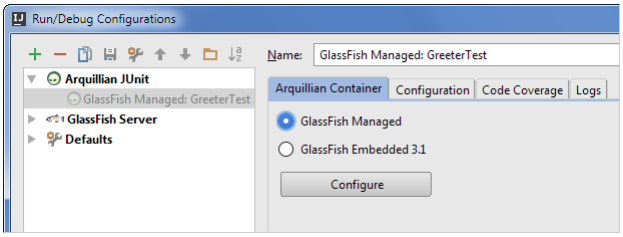
3. In the Arquillian Containers dialog, click + and select Manual container configuration.
4. Change the name of the configuration (e.g. to `GlassFish Managed`).
5. Under Dependencies, click + and select Add maven dependency.
6. In the Download Library From Maven Repository dialog, type `arquillian-glassfish-managed-3.1` and click 🔍. Then select org.jboss.arquillian.container:arquillian-glassfish-managed-3.1:1.0.0.CR4 from the list.
   Select the Download to check box and click OK.



   At this step, your Arquillian Containers dialog should look something like this:



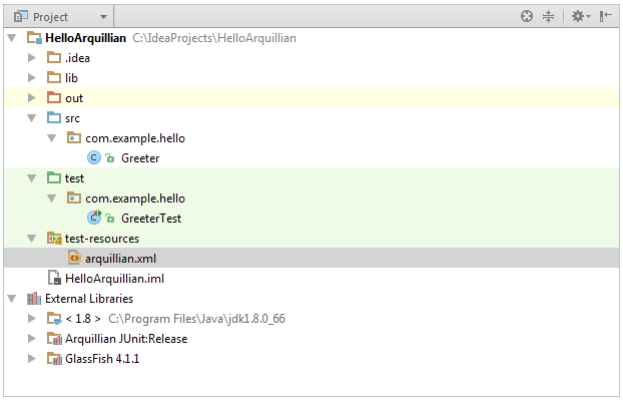
   Click OK.
7. In the Run/Debug Configurations dialog, select GlassFish Managed and click OK.



Creating the arquillian.xml configuration file

To be able to run an Arquillian test in a managed container, the container adapter needs to know the container location. So let's create the `arquillian.xml` configuration file with the necessary info.

1. In the project root folder, create the folder `test-resources`.
2. Right-click the new folder, point to Mark Directory As and select Test Resources Root.
3. In the test-resources folder, create a new file `arquillian.xml`.



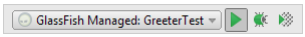
4. Copy the following into the file:

```
<?xml version="1.0"?>
                    <arquillian xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns="http://jboss.org/schema/arquillian"
                    xsi:schemaLocation="http://jboss.org/schema/arquillian
                        http://jboss.org/schema/arquillian/arquillian_1_0.xsd">

                    <container qualifier="glassfish" default="true">
                    <configuration>
                    <property name="glassFishHome">C:\GlassFish\glassfish4</property>
                    </configuration>
                    </container>

                    </arquillian>
```

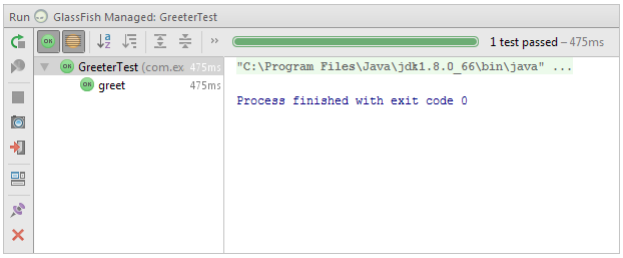
Use your actual path to the GlassFish Server installation folder in place of `C:\GlassFish\glassfish4`.

Running the test in a managed container

1. To the right of the run configuration selector, click ▶.



The Run tool window opens and the test result is shown there.



2. Close the tool window ( ).

Modifying arquillian.xml

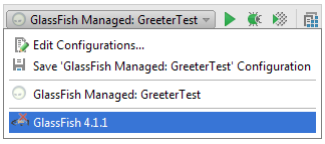
Sometimes, you need to deploy your test to a container that is already running. In such cases, you can just change the container configuration file a little and continue using the managed container adapter.

Add the following to `arquillian.xml` :

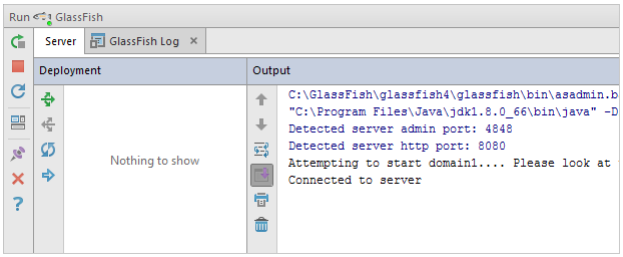
```
<property name="allowConnectingToRunningServer">true</property>
```

Running the test: deploying to a running server

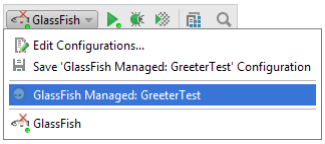
1. Start GlassFish Server: select GlassFish and click .



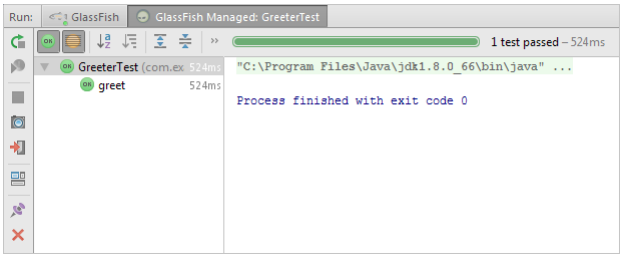
When the server is started, you'll see something like this in the Run tool window.



2. Select GlassFish Managed: GreeterTest and click .



After some time, the test result is shown in the Run tool window.



AspectJ

In this section:

- AspectJ
  - Introduction
  - Overview of AspectJ support
  - Overview of using AspectJ support
- Enabling AspectJ Support Plugins
- Creating a Library for aspectjrt.jar
- Creating Aspects
- Using the Push ITDs In Refactoring
- Using the AspectJ Compiler (ajc)

Introduction

AspectJ support in IntelliJ IDEA Ultimate is based on the following plugins that are bundled with the IDE:

- Spring AOP/@AspectJ
- AspectJ Support

An AspectJ facet which you can add to your Java modules lets you fine-tune the use of the AspectJ compiler at a module level.

Overview of AspectJ support

AspectJ support in IntelliJ IDEA Ultimate includes:

- Ability to create aspects in two forms: as `.aj` files and `.java` files containing classes annotated with `@Aspect`.
- Coding assistance (including code completion) when writing aspect code. For annotation-style aspects, the coding assistance is provided in full; for code-style aspects, the assistance is limited to inter-type declarations .
- Ability to perform basic aspect refactorings such as Rename and Move , and also the Push ITDs In refactoring for inter-type declarations.
- Integration with the AcpectJ compiler `ajc` which you can run right from the IDE. (This compiler is part of the AspectJ distribution which you can download from the AspectJ website .)
- Ability to configure `ajc` at the project level with an option of fine-tuning its use at the level of individual modules.

Overview of using AspectJ support

The tasks that are specific to AspectJ are outlined below. For general instructions, see How-to .

1. Make sure that:
   - You are using the Ultimate Edition of IntelliJ IDEA. AspectJ is not supported in the Community Edition.
   - The Spring AOP/@AspectJ and the AspectJ Support plugins are enabled. See Enabling AspectJ Support Plugins .
2. Download and install AspectJ.
3. Create a library containing `aspectjrt.jar` and add this library to dependencies of the modules in which you are going to develop your aspects. Once you have installed AspectJ, you can find `aspectjrt.jar` in `<AspectJ installation directory>\lib`. See Creating a Library for aspectjrt.jar .
4. Create aspect files and develop the code. Note that code- and annotation-style aspects are supported.
5. If necessary, refactor the aspect code .
6. To use the AspectJ compiler, configure the compiler settings . The compiler (`ajc`) is in `aspectjtools.jar` which is located in `<AspectJ installation directory>\lib`.
7. To fine-tune the use of `ajc` at the level of individual modules, add AspectJ facets to corresponding modules and adjust the facet settings accordingly.

Enabling AspectJ Support Plugins

To be able to use the AspectJ support in IntelliJ IDEA Ultimate, you have to make sure that the following plugins are enabled:

- Spring AOP/@AspectJ
- AspectJ Support

In addition to that, you should also create a library for aspectjrt.jar .

Making sure that the necessary plugins are enabled

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, in the search box, type `plugins`. Then, select Plugins in the pane below the search box.
3. On the Plugins page that opens in the right-hand part of the dialog, make sure that All plugins is selected in the Show box, and type `aspectj` in the search field.
4. Make sure that the check boxes to the right of AspectJ Support and Spring AOP/@AspectJ are selected.
5. Click OK in the Settings dialog.
6. If asked, restart IntelliJ IDEA.

Creating a Library for aspectjrt.jar

To be able to use the AspectJ support, in addition to enabling AspectJ support plugins , you should create a library containing `aspectjrt.jar` and add this library to dependencies of the modules in which you are going to develop your aspects.

`aspectjrt.jar` is included in the AspectJ distribution which you can download from the AspectJ website .

Creating a library for aspectjrt.jar and adding it to module dependencies

By the time you start to perform this task, you must have already downloaded and installed AspectJ.

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. Depending on whether you want to create the new library at the project or the IDE level, select Libraries or Global Libraries.
3. Click + and select Java.
4. In the dialog that opens , find and select `aspectjrt.jar`. (This file is located in `<AspectJ installation directory>\lib`.)
5. In the Choose Modules dialog, select the modules in which you want to use this library and click OK.
   As a result, the library is added to dependencies of the corresponding modules.
6. Click OK in the Project Structure dialog.

Creating Aspects

IntelliJ IDEA lets you create aspects in two forms: as `.aj` files and `.java` files containing classes annotated with `@Aspect`.

Note that by the time you start to create your aspects, you must have already enabled the AspectJ support plugins and defined aspectjrt.jar as a library.

Creating an aspect

1. In the Project tool window, do one of the following:
   - Select your source directory or the package in which you want to create a new aspect, and select File | New.
   - Select the source directory or package, and press `Alt+Insert`.
   - Right-click the source directory or package to open the context menu, and select New.
2. Select Aspect.
3. In the dialog that opens:
   1. Specify the name of the aspect in the Name field.
   2. Select one of the following options from the Kind list:
      - Aspect to create a `.aj` file with the specified name.
      - @Aspect to create a `.java` file containing a class annotated with `@Aspect`.
   3. Click OK.

Using the Push ITDs In Refactoring

For aspects, IntelliJ IDEA supports most of the basic refactorings such as Move and Rename . In addition to that, you can also perform the Push ITDs In refactoring for inter-type declarations .

The Push ITDs In refactoring lets you move the definitions of fields and methods from aspects to corresponding classes. (The ITDs in the refactoring name stands for Inter-Type Declarations.)

The refactoring may be performed for an individual field or method, for an aspect as a whole, or for all aspects in a package.

If when performing the refactoring certain aspects become "empty" (this may be the case when the corresponding aspects contain nothing but inter-type declarations), you can select to automatically delete all such aspects.

- Examples
- Performing the Push ITDs In refactoring

Examples

The following table shows examples of the Push ITDs In refactoring.

In the first example, the declaration of the field `closed` is moved from the aspect `MyAspect` to the class `Account`.

In the second example, the method `close()` is also moved from `MyAspect` to `Account`.

Note that both these declarations (that is, the declarations of `closed` and `close()`) may be moved to the class `Account` at once if the refactoring is performed for the aspect `MyAspect` as a whole.

Before **After**

```
aspect myAspect {
                    boolean Account.closed = false;
                    void Account.close()
                    {
                    closed = true;
                    }
                    // some code here
                    }
                    class Account {
                    // some code here

                    }
```

```
aspect myAspect {

                    void Account.close()
                    {
                    closed = true;
                    }
                    // some code here
                    }
                    class Account {
                    // some code here
                    boolean closed = false;
                    }
```

```
aspect MyAspect {
                    void Account.close()
                    {
                    closed = true;
                    }
                    // some code here
                    }
                    class Account {
                    // some code here
                    boolean closed = false;



                    }
```

```
aspect MyAspect {



                    // some code here
                    }
                    class Account {
                    // some code here
                    boolean closed = false;
                    void close() {
                    closed = true;
                    }
                    }
```

Performing the Push ITDs In refactoring

1. Depending on the intended scope of the refactoring:
   - To perform the refactoring for an individual field or method, open the aspect of interest in the editor and place the cursor within the declaration of the field or method.
   - To perform the refactoring for an aspect as a whole, select the aspect of interest in the Project tool window. Alternatively, open the aspect in the editor and place the cursor somewhere outside of individual inter-type declarations (e.g. within the declaration of the aspect).
   - To perform the refactoring for all aspects in a package, select the package in the Project tool window.
2. Select Refactor | Push ITDs In in the main or the context menu.
3. In the Push Inter-Type Declarations In dialog that opens:
   1. Select or clear the Delete empty aspects check box.
   2. Click Refactor to perform the refactoring right away, or Preview to be able to study the expected changes prior to actually performing the refactoring.
4. If at the previous step you clicked Preview, the Find tool window opens showing the inter-type declarations that are going to be affected. If happy with the expected result, click Do Refactor.

Using the AspectJ Compiler (ajc)

By default, IntelliJ IDEA uses the `javac` compiler. To use the AspectJ compiler ajc (instead of or in combination with `javac`), you should make changes to corresponding IDE settings.

The `ajc` settings specified at the project level can be fine-tuned at the level of individual modules. AspectJ facets associated with the modules are used for that purpose.

Note that `ajc` is not bundled with IntelliJ IDEA and should be downloaded separately.

`ajc` is available as part of the AspectJ distribution which you can download from the AspectJ website.

- Optimizing compilation performance: Using ajc in combination with javac
- Controlling the ajc aspectpath
- Selecting ajc as the project compiler and specifying its settings
- Fine-tuning the use of ajc at a module level

Optimizing compilation performance: Using ajc in combination with javac

IntelliJ IDEA lets you use `ajc` in combination with `javac` without the need to switch the compilers in the IDE settings.

First of all, you should select `ajc` as your project compiler (the Use compiler field on the Java Compiler page).

If you want `javac` to be also used, turn on the Delegate to Javac option. If this option in on, the modules without aspects are compiled with `javac` (which is, generally, faster), and the modules that contain aspects are compiled with `ajc`. (If this option is off, `ajc` is used for all the modules in your project.)

You can fine-tune the distribution of tasks between the compilers (`ajc` and `javac`) at the level of individual modules. For modules that contain aspects only in the form of `@Aspect`-annotated Java classes (in `.java` files), you can specify that `ajc` should be used only for post-compile weaving. If you do so, `javac` will be used to compile all the source files, and then `ajc` will be applied to the compiled class files for weaving. As a result, the overall process (compilation + weaving), will take less time.

Provided that the Delegate to Javac option is on, the post-compile weaving mode for `ajc` is enabled by turning on the corresponding option in an AspectJ facet associated with a module.

Note that you shouldn't turn on this option for modules that contain code-style aspects (ones defined in `.aj` files).

Controlling the ajc aspectpath

You can control the ajc `aspectpath` option separately for each of your modules, see Fine-tuning the use of ajc at a module level.

The module `aspectpath` may be set automatically as a result of importing an appropriately configured Maven project into IntelliJ IDEA.

Selecting ajc as the project compiler and specifying its settings

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, in the search box, type `compiler`. Then, select Java Compiler in the pane below the search box.
3. On the Compiler > Java Compiler page that opens in the right-hand part of the dialog:
   1. Select Ajc from the Use compiler list.
   2. If necessary, specify the bytecode versions. (Roughly, these are the minimum target JVM versions.)
   3. Specify the path to the compiler in the Path to Ajc compiler field.
      You can type the path in the field, or click … and select the necessary file in the corresponding dialog.

      (The file that you want is `aspectjtools.jar` which is located in `<AspectJ installation directory>\lib`.)
   4. If necessary, specify the command-line options to be passed to the compiler in the Command line parameters field.
      You can type the parameters right in the field, or click to open the Command line parameters dialog where the text entry area is larger.
   5. Click Test to check if the specified settings are correct.
   6. If you want to use `javac` to compile the modules that contain no aspects, select the Delegate to Javac check box.
      Note that the distribution of tasks between `ajc` and `javac` can be fine-tuned at the level of individual modules.
4. Click OK in the Settings dialog.

Fine-tuning the use of ajc at a module level

To be able to fine-tune the use of `ajc` at a module level, you should add an AspectJ facet to corresponding module or modules.

Once you've done that, you can specify that `ajc` should be used only for post-compile weaving. You can also specify the `aspectpath` for the module.

1. Add an AspectJ facet to the module of interest. For corresponding instructions, see Adding and Removing Facets.
2. On the page that opens in the right-hand part of the Project Structure dialog, under Compiler, specify the settings as necessary (see below).
3. If you want to use `ajc` only to weave the compiled class files (the source code in this case is compiled with `javac`), select the Post-compile weave mode check box.
   IMPORTANT: Don't select this check box if the module contains `.aj` files.
4. To form the `aspectpath` for the module:
   - Use + (`Alt+Insert`) to add libraries and other modules. Select the necessary libraries and modules in the dialog that opens. (To choose from, dependencies of the module are suggested.)
   - Use − (`Alt+Delete`) to remove the selected items from the list.
   - Use ↑ (`Alt+Up`) to move the selected item one line up in the list.
   - Use ↓ (`Alt+Down`) to move the selected item one line down in the list.
5. Click OK in the Project Structure dialog.

Build Tools

IntelliJ IDEA supports the following build scripting tools:

- Ant
- Gant
- Gradle
- Maven

Ant

Ant is a flexible, platform-independent build tool from Apache Ant Project . IntelliJ IDEA integrates with Ant to provide a comprehensive build process, that includes compilation, packaging with the documentation and source code, committing to version control and much more.

Ant integration is shipped with IntelliJ IDEA, and you do not need to perform any additional actions to install it. However, it is also possible to use the other Ant installations.

Ant support in IntelliJ IDEA imposes certain prerequisites, and includes the following features:

- Ant
  - Prerequisites
  - Dedicated tool window
  - Ant build files
  - Ant build targets
  - Coding assistance
  - Path-like structures
- Creating Ant Build File
- Generating Ant Build File
- Adding Build File to Project
- Controlling Behavior of Ant Script with Build File Properties
- Running the Build
- Working with Ant Properties File

Prerequisites

Before you start working with Ant, make sure that the Ant support plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Dedicated tool window

Ant Build tool window enables adding Ant build scripts to IntelliJ IDEA project, control behavior of the build, and execute Ant build targets.

Ant build files

Ant works with the XML build file. Normally, the name of the build file is `build.xml` . Build file describes the steps, or build targets, required to build a project. The root element of the build file is `<project>` . IntelliJ IDEA makes it possible to work with existing build files, create new build files from scratch, or generate them automatically.

IntelliJ IDEA is aware of specific Ant syntax. However, you have to let IntelliJ IDEA know that a certain XML file is in fact an Ant build file. To be recognized as a build file and enable all advanced editing features, an Ant build file should meet at least one of the following requirements:

- The file should be properly added to the project .
- The `<project>` root element should have `default` attribute.

Otherwise such files are treated as regular XML files with basic editing support. Once a build file is added to a project, it can be used to run the build and modify its properties.

Ant build targets

A build target is identified with a unique name and defines a procedure that should be executed to accomplish a certain task, for example, create a JAR file, or generate API documentation. A target specified in the `default` attribute of the `<project>` element is considered the default target, which is executed when no other target is specified. This target is called the primary target, and is marked with bold font in the Ant Build tool window .

Coding assistance

When editing Ant build files in IntelliJ IDEA, you can enjoy the following advanced editing features:

- Syntax highlighting.
- Code completion. In particular, code completion is provided for the properties of the File type.
- Navigating to declaration `Ctrl+B` .
- Using Structure view.
- Rename refactoring.
- Code folding.
- Reformatting.
- Validation.
- Viewing parameter information `Ctrl+P` .
- Viewing quick info `N/A` . In particular, if classpath is defined as a path-like structures, the View Quick Info command for the `fileset` or `dirset` directives displays the actual files and directories on the disk, to which these directives are resolved.

Path-like structures

IntelliJ IDEA enables using path-like structures in the task definitions. If a classpath is defined as a path-like structure, the paths in the `fileset` and `dirset` directives are resolved into the actual files and directories on the disk. All JARs, required for performing the task, should be placed to the same place that contains the JAR with task definitions.

Creating Ant Build File

This section describes how to create the Ant build file manually.

IntelliJ IDEA provides a framework for editing build files, but it is the developer's responsibility to populate the build file with targets.

To create Ant build file

1. In the Project window, select the directory, where the build file should be created.
2. Right-click the directory and choose New | File on the context menu, or press `Alt+Insert`.
3. In the New File dialog, specify the name of the new file with `.xml` extension, for example, `build.xml`. The new file opens in the editor.
4. Specify the build targets. You can use the path-like structures in the `fileset` or `dirset` directives to define
   You have to enter at least a root tag in the `build.xml` file. Otherwise the file will not be added to the Ant Build tool window.
5. Add the build file to the project .

Generating Ant Build File

IntelliJ IDEA provides the possibility to automatically generate Ant build files for a project, or several build files per each module comprising the project. Once the build file is generated, you can add it to the Ant Build tool window . You can edit and manage such build file as a regular file.

To generate Ant build file

1. On the main menu, choose Build | Generate Ant Build.
2. In the Generate Ant Build dialog , specify the generation options.
   - Choose to generate a single-file or multiple-file build.
   - Choose to preserve or overwrite the previously generated files.

   Refer to the topic Generate Ant Build Dialog for detailed description of controls.
3. Click OK.

Adding Build File to Project

A build file should be added to a project to enable running the build, filtering targets, or defining properties. One project can have several build files. The build files are grouped in the Ant Build tool window by names, specified in the root element of each build file, for example `<project name="acme">`.

When a build file is generated, it is automatically added to the project. If you create a build file manually, or reuse an existing one, you might need to add it to the Ant Build Tool Window.

To add a build file to a project

1. Open the Ant Build Tool Window. To do that, choose View | Ant Build on the main menu, or click the Ant Build button in the right toolbar of the tool windows.
2. In the Ant Build tool window, click +.
3. In the Select Path dialog, navigate to the desired `build.xml` file, and click OK.
   A build file should have at least a root element to be added to the Ant Build tool window.

Controlling Behavior of Ant Script with Build File Properties

You can control the way IntelliJ IDEA executes Ant scripts. If you want to perform execution in the background, change amount of memory allocated to the build process, or set other execution options, you'll need to open the Build File Properties dialog. This section describes how to do it.

To open the Build File properties dialog

1. In the Ant Build Tool Window , select the desired build file.
2. Do one of the following:
   - On the context menu of the selection, choose Properties.
   - Click button on the Ant Build toolbar.
   - Press `Alt+Enter`.

For specific tasks refer to the following procedures:

- Increasing memory heap .
- Executing Build File In Background .
- Defining Runtime Properties .
- Defining Additional Ant Classpath .
- Defining Ant Execution Options .
- Defining Ant Filters .

Defining Additional Ant Classpath

Ant build scripts require classpaths that are independent from IntelliJ IDEA, or additional libraries for proper functioning. This section describes how to add directories and archives to the classpath, and change the order in which Ant loads the resources.

To configure Ant classpaths

1. Open the Build File Properties dialog .
2. Select the Additional Classpath tab.
3. Click Add, and in the Select Path dialog box select an archive or a directory to be added to the classpath.
   If you want to add the contents of a whole directory, click the Add All In Directory button, and select the desired directory in the Select Path dialog box.
4. Use Move Up and Move Down buttons to change the order of classpath entries.

Defining Ant Execution Options

Using the Build File Properties dialog, you can control how IntelliJ IDEA launches the Ant build process. In particular, you can define which version of Ant should be used, add command line arguments, and specify the SDK to be used for running Ant.

To define execution options

1. Open the Build File Properties dialog box.
2. Select the Execution tab.
3. In the Run With Ant section, specify whether you want to use project default or custom Ant version. If you want to use custom Ant version rather than the bundled one, select it from the drop-down list, or click the ellipsis button and configure Ant by adding classpaths .
4. In the Ant command line field, type the command line arguments, using the standard Ant syntax: precede arguments with dashes, and separate with spaces. For the lengthy command lines, click and type the text in the Ant Command Line dialog.
5. In the Run under JDK field, specify a SDK to be used for Ant process: select the SDK from the list, or click the ellipsis button and configure an individual SDK .

Defining Ant Filters

By default, the Ant Build Tool Window shows all targets of a build file. Filtering enables you to show or hide targets as desired. The default filter displays the primary targets only. You can change this behavior and configure your own filter with custom set of targets that should be visible when the filter is applied.

To configure custom filter for build targets

1. Open the Build File Properties dialog box.
2. In the Filters tab, clear the check boxes next to the build targets you want to hide, and select the check boxes next to the targets you want to show.
   Now, when you apply filter by pressing the filter button , only the targets selected in the Filters list will be displayed.

Defining Runtime Properties

Use the Properties tab of the Build File Properties Dialog to pass properties to the build script at runtime. The specified values are equivalent to those defined after the `-D` option of the command line launcher.

In addition to plain values, you can use macros that are evaluated at runtime. Such macros are helpful, when you have to pass specific paths and other varying information to the build script. Macros are character strings surrounded with dollar signs. Build File Properties dialog provides the complete list of macros, available for the selected build file, with the previews that show, how these macros will be evaluated at runtime.

To define the runtime properties

1. Open the Build File Properties dialog .
2. In the Properties tab, click Add.
3. In the Name column, type the property name.
4. In the Value column, type the desired value. If you use a macro as the property value, type the name of the desired macro. If you don't know the name, click the + button, select the desired macro from the Macros dialog box, and click OK.
5. Use Add and Remove buttons to make up the complete list of properties.

Executing Build File in Background

By default, during the build process Ant displays a modal dialog that shows the progress of the build. It is possible to execute the build in the background, and use this time to work on other things.

Warning! Background execution may slow down performance.

To enable background execution of a build file

1. Open the Build File Properties dialog box.
2. Check the Make build in background option.

Increasing Memory Heap

To prevent the build process from running out of memory, you can increase the amount of memory allocated to the process. By default, the memory heap is 128 MB, but for large projects you may need more.

In this section:

– Increasing memory heap of the build process
– Important notes

Increasing memory heap of the build process

To increase a memory heap

1. Open the Build File Properties dialog box.
2. In the Maximum heap size field, type the required amount of memory.

Important notes

Please note the following:

The memory heap of the build process is independent of IntelliJ IDEA memory heap, and is released after the build process is complete.

The memory heap available to IntelliJ IDEA may be changed by editing the corresponding VM options.

To avoid editing files in the IntelliJ IDEA installation folder, one should:

1. Do one of the following:
   – Use the main menu command Help | Edit Custom VM Options to create a copy of the `idea.vmoptions` file in the user home. Refer to Edit Custom VM Options for details.
   – Copy the existing file from the IntelliJ IDEA installation folder somewhere and save the path to this location in the environment variable `IDEA_VM_OPTIONS`.
   – Copy the existing file `<IntelliJ IDEA installation folder>/bin/idea$BITS.vmoptions` from the IntelliJ IDEA installation folder into the location under user home.
2. Edit this file in the new location.

If `IDEA_VM_OPTIONS` environment variable is defined, or the `*.vmoptions` file exists, then this file is used instead of the one located in the IntelliJ IDEA installation folder.

Running the Build

This section describes how to prepare and perform Ant build:

- Assign a keyboard shortcut to an Ant target .
- Configure triggers for the Ant targets .
- Execute an Ant target .

Associating Ant Target with Keyboard Shortcut

You can associate a build target with a keyboard shortcut and execute commonly-used targets with a single key-stroke. If an Ant build file is added to the project, its targets appear under the Ant Targets node in the Keymap dialog box.

To associate a keyboard shortcut with a build target

1. In the Ant Build Tool Window , right-click the desired build target.
2. On the context menu, choose Assign Shortcut. Keymap dialog is opened.
3. Configure keymap, as described in the section Configuring Keyboard Shortcuts .

Configuring Triggers for Ant Build Target

You can configure triggers that run Ant targets before or after certain events, for example, compiling, running, or testing. These triggers are called compilation trigger and execution trigger respectively.

To set up a compilation trigger

1. On the Ant Build Tool Window , right-click the desired build target.
2. On the context menu of the target, choose Execute on.
3. On the submenu, choose Before Compilation or After Compilation, as required. Note that you can set up both options.

To set up an execution trigger

1. On the Ant Build tool window, right-click the desired build target.
2. On the context menu of the target, choose Execute on.
3. On the submenu, choose Before Run/Debug. Execute Target Before Run/Debug dialog appears, presenting the list of possible execution types.
4. In the dialog, select check boxes next to the types of execution that should trigger the selected Ant target, and click OK.

Executing Ant Target

With IntelliJ IDEA you can run the build targets, review results of compilation and build, and navigate to the point of origin of each error.

Build targets can be executed from:

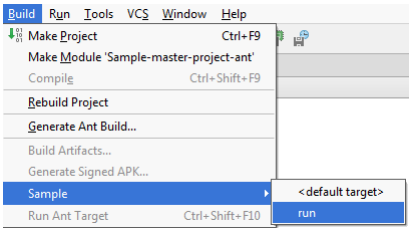
- The Ant Build tool window .
- The main Build menu that contains the list of all targets defined in the build files, added to a project.

To execute a build target from the Ant Build tool window

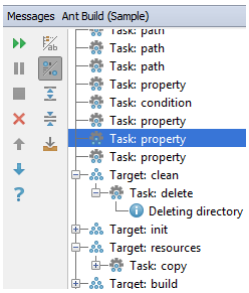
1. In the Ant Build Tool Window , select the desired target.
2. Do one of the following:
   - On the context menu of the selected target, choose Run target (or Run build, if you execute the entire build file).
   - Click the button on the toolbar of the Ant Build Tool Window .
   - Double-click the selected target.

To execute a build target from the main menu

1. On the main menu, click Build.
2. Click the submenu that corresponds to the desired build file.
3. Select the desired build target.



In both cases, results are displayed in the Messages tool window :



Creating Meta Target

IntelliJ IDEA lets you create a meta target for your Ant build file. A meta target can contain several different targets of your choosing. Those targets will be executed in the specified order. In this case, you don't need to change the Ant build script itself.

You can treat your meta target as a regular Ant target and assign shortcuts to it, configure triggers for running the target, etc.

To create a Meta Target

1. In the Ant Build Tool Window , press `Ctrl` and simultaneously click the desired Ant targets to mark them as selected.
2. Right-click on one of the targets to open a context menu.
3. From the context menu, select Create Meta Target.
   (If you need to remove the created meta target, select Remove Meta Target.)
4. In the Create Meta Target dialog, check the order of your targets and the name of your meta target. If you need, you can edit the specified information accordingly. Click OK.

Working with Ant Properties File

Ant properties files let you move properties out of you build.xml file. Once you create a property file, IntelliJ IDEA recognizes the file and tries to match key-value-pairs from that file against the Ant or Java files. If the match is found, IntelliJ IDEA lets you use a link between properties of those files, ensures the correct rename of the keys and updates the links accordingly.

You create `.properties` file for all the properties that are defined outside the build file. For example, properties that you define in build settings. IntelliJ IDEA keeps those properties in one place and you don't need to edit the generated build file.

IntelliJ IDEA generates `.properties` file automatically if you generate the Ant build file . If you create build.xml manually then to use a `.properties` file in your Ant build.xml file do the following:

– Include a property in your build.xml file with the name of the properties file specified by the file attribute. For example, `property file="build.properties"` .

Gant

In this section:

- Gant
  - Prerequisite
  - Gant support
- Running Gant Targets
- Adding Gant Scripts

Prerequisite

Before you start working with Gant files, make sure that Gant is downloaded and installed on your computer.

Gant support

IntelliJ IDEA provides Gant support. IntelliJ IDEA recognizes `*.gant` files, and allows editing them.

Gant files are marked with icon.

Gant support includes:

- Code completion for Gant tasks.
- Execution of a whole script.
- Selective run of the individual targets.
- Groovy script run/debug configuration .

Running Gant Targets

With IntelliJ IDEA, it is possible to run an entire Gant script, or each target separately, using the temporary run/debug configurations. Temporary run configuration for a script or target can be saved as a permanent one.

Results display in the Run tool window .

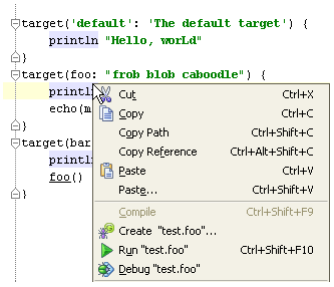
To run a Gant script

– Select the desired Gant script, and run it with the default run/debug configuration.

To run a Gant target

1. Open the desired Gant script in the editor, and place the caret at the Gant target to be executed.

2. On the context menu, choose Run "<script><target>"



Adding Gant Scripts

IntelliJ IDEA lets you add Gant scripts for Groovy, Grails and Griffon projects.

Note For Groovy projects you need to specify the location of the Gant home directory in Project Settings | Gant before adding the Gant scripts.

To add Gant Scripts

1. In the Project tool window, right click the directory where you want to create a Gant script file.
2. Select New | Gant Script from the context menu.
3. In the New Gant Script dialog, specify the name of the script and click OK.
4. IntelliJ IDEA creates a Gant file with the default target.

Gradle

IntelliJ IDEA supports Gradle . In this section:

- Gradle
  - Prerequisites
  - Gradle Support
- Getting Started with Gradle
- Creating a Gradle Project
- Creating Gradle Project with Grails Integration
- Importing Project from Gradle Model
- Working with Gradle Projects
- Gradle Selective Importing
- Synchronizing Changes in Gradle Project and IntelliJ IDEA Project
- Working with Gradle Tasks in Gradle Tool Window
- Associating Gradle Tasks with Keyboard Shortcuts

Prerequisites

Before you start working with Gradle files, make sure that:

- Gradle is downloaded and installed on your computer.
- IntelliJ IDEA supports Gradle SDK version 1.2. and later.
- Gradle plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Gradle Support

Gradle files are marked with icon.

Gradle support includes:

- ability to deduce Gradle home from the environment variables, or configure Gradle home in IntelliJ IDEA.
- ability to edit `.gradle` files. So doing, the initial coding assistance, provided by IntelliJ IDEA, is the same as for Groovy .
- ability to import projects from the existing Gradle models. So doing, IntelliJ IDEA downloads all the necessary dependencies.
- Dedicated Gradle tool window .
- ability to synchronize structures of Gradle and IntelliJ IDEA projects.
- Running Gradle scripts with the Groovy run/debug configuration .

Getting Started with Gradle

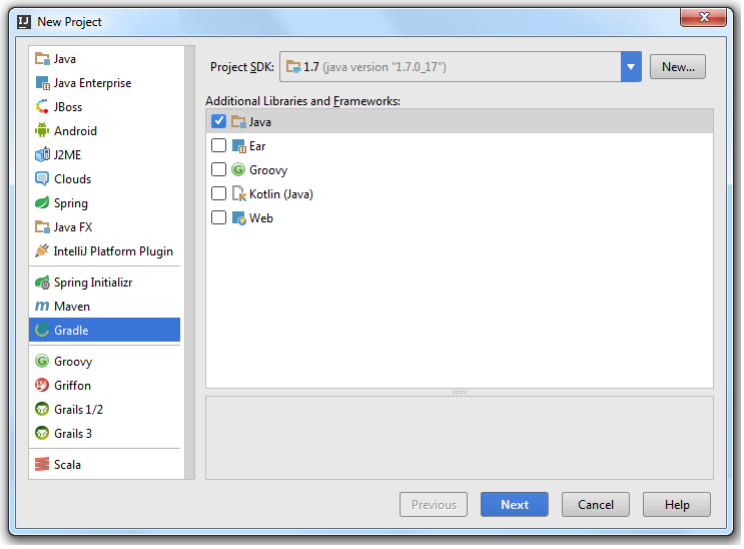
IntelliJ IDEA supports Gradle plugin version 1.8 and later and lets you take full advantage of the Gradle integration.

- Creating Gradle Project
- Importing Project from Gradle Model
- Exploring Gradle Project Structure
- Working with Gradle Tool Window

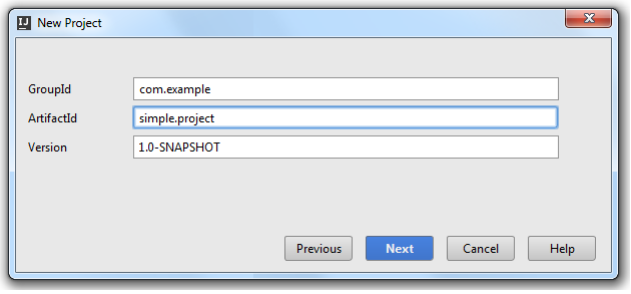
Creating Gradle Project

1. Open Project Wizard, in the left-hand pane select Gradle.
2. In the right-hand pane, specify the following options:
   - Project SDK - specify your project SDK (JDK).

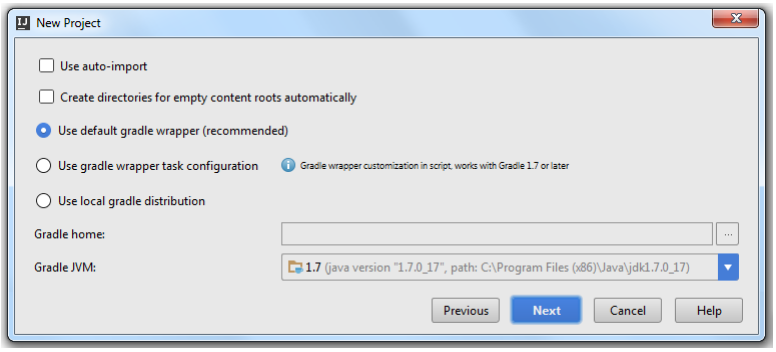
     If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select the installation folder of the desired JDK.
   - Additional Libraries and Frameworks - use this area to select an additional framework or a library for your Gradle project.



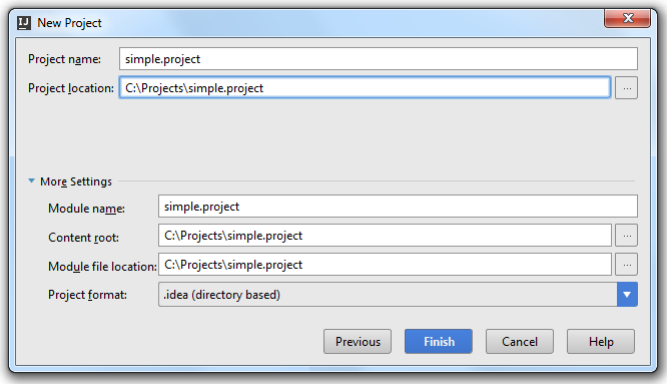
   Click Next.
3. On the next page of the wizard, specify the following settings:
   - GroupId - specify `groupId` of the new project, which will be added to the `build.gradle` file.
   - ArtifactId - specify `artifactId` of the new project.
   - Version - specify `version` of the new project, which will be added to the `build.gradle` file.



   Click Next.
4. On the next page of the wizard, specify the Gradle settings and click Next.



5. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location . Click Finish.



IntelliJ IDEA creates a project with `build.gradle` file.

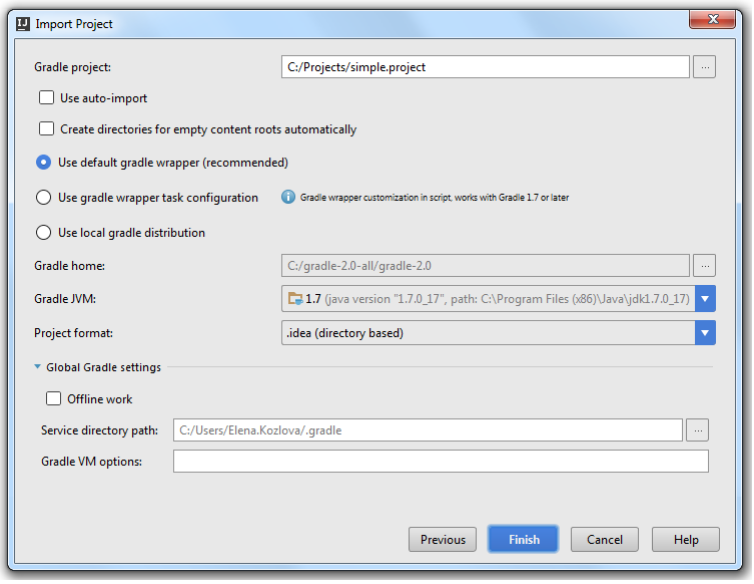
Note You can also create a Gradle project with the Grails integration. For more information, please see Creating Gradle Project with Grails Integration .

Importing Project from Gradle Model

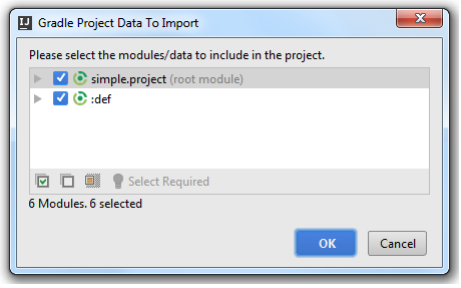
1. On the main menu, select File | Open, from the dialog that opens select `build.gradle` file and click OK.

   IntelliJ IDEA will import Gradle project even if the project was not opened or imported before.

   Alternatively, you can select File | New | Project from Existing Sources and from the dialog that opens select the Gradle project that you want to import.
2. On the next page, select the model from which you want to import and click Next. If IntelliJ IDEA guessed what you are importing, then this page will be skipped.
3. On the next page of the wizard, specify the Gradle project settings and global Gradle settings and click Finish.



If you import a multi-module project, IntelliJ IDEA displays the Project Data to Import dialog where you can select modules that you want to include in the project.

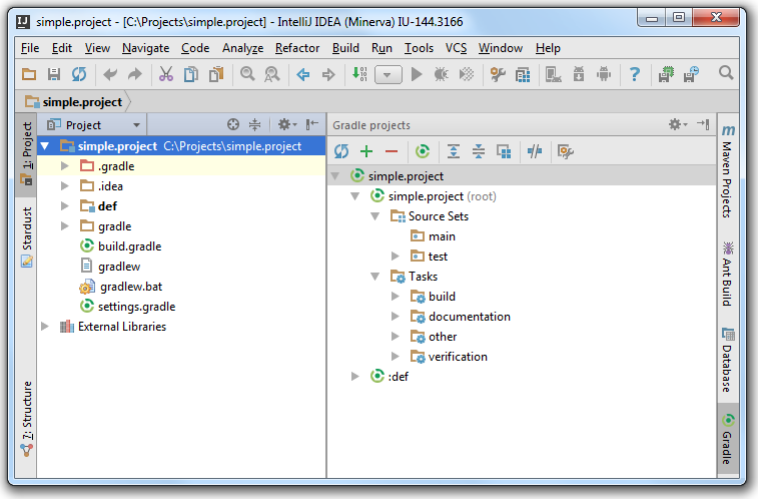


Note For more information on the Gradle selective importing, please see Gradle Selective Importing .

Exploring Gradle Project Structure

IntelliJ IDEA lets you view your Gradle project from the two following points:

- Project View
- Gradle tool window

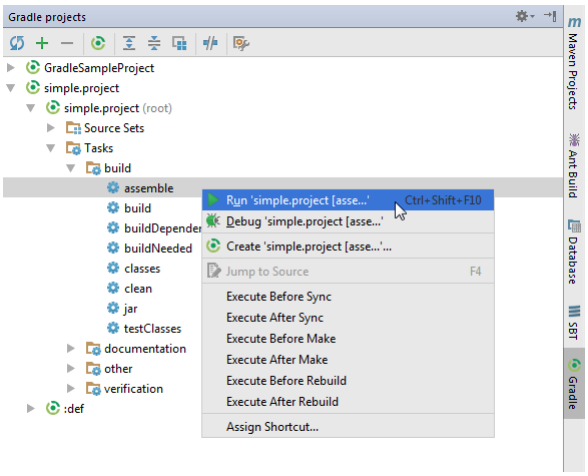


You can make changes to the structure of the original Gradle project and to the resulting IntelliJ IDEA project, synchronize structures of the two projects, and explore differences.

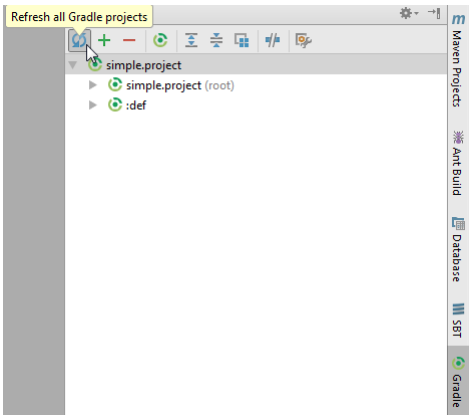
Working with Gradle Tool Window

IntelliJ IDEA lets you perform the following actions in the Gradle tool window:

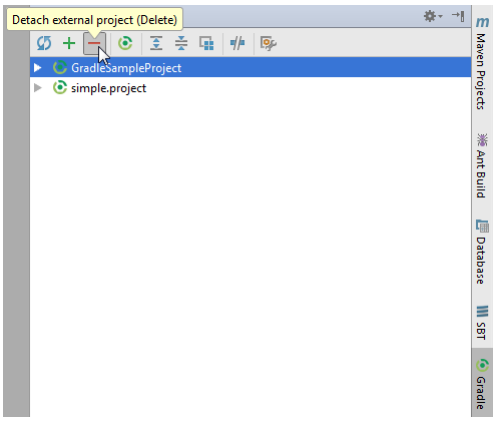
– run, debug or create the Run/Debug configuration for Gradle tasks



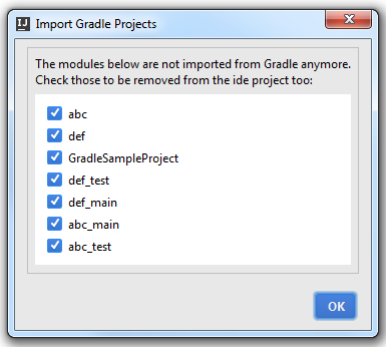
– sync your projects



– add and remove linked projects



If you have a multi-module project, IntelliJ IDEA displays the Import Gradle Projects dialog where you can select modules that you want to remove from IntelliJ IDEA project.



Creating a Gradle Project

You can create a new Gradle project or add a new Gradle module to an existing project.

To create a Gradle project

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Gradle. In the right-hand part of the page, specify the JDK that you are going to use.
   If you need, specify additional libraries and frameworks.

   Click Next.

3. On the next page of the wizard, specify the following settings:
   - Add as module to - this field appears when you add a new module to the existing project. By default, the field displays the the name of your project. Click to select a different name.
   - GroupId - specify `groupId` of the new project, which will be added to the `build.gradle` file. If a parent Gradle project is specified, this coordinate can be inherited from the parent. To do that, select the Inherit check box.
   - ArtifactId - specify `artifactId` of the new project, which will be added to the `build.gradle` file.
   - Version - specify `version` of the new project, which will be added to the `build.gradle` file. If a parent Gradle project is specified, this coordinate can be inherited from the parent. To do that, select the Inherit check box.

   Click Next.

4. On the next page of the wizard specify the Gradle settings .
   Click Next.

5. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Creating Gradle Project with Grails Integration

IntelliJ IDEA lets you create a Gradle project with Grails integration using Gradle-Grails plugin . The plugin lets you build Grails applications using Gradle.

To create Gradle Project with Grails integration

1. Create a simple Gradle project .
2. Open `build.gradle` file.
3. From the Gradle-Grails plugin Website copy the content of the `build.gradle` example and paste it to the opened build file inside your IDE.

   Note Make sure that the Gradle, Grails and Groovy plugin versions are compatible between each other.
4. In the Gradle tool window , click icon to refresh your Gradle project.
   The list of Grails tasks are added to the All tasks area in the Gradle Tool Window.
5. From the All tasks list, run init task.
   The task creates a Grails application. The Grails View tool window becomes available and you can start working with Grails.

   For more information on working with Grails, see Grails procedures .

To import an external Gradle model into IntelliJ IDEA, follow these steps

1. If no project is currently open in IntelliJ IDEA, click Import Project on the Welcome screen . Otherwise, select File | New | Project from Existing Sources.
2. In the dialog that opens , select the directory that contains the project to be imported, or a file that contains an appropriate project description. Click OK.
3. On the first page of the Import Project wizard, select Gradle, and click Next. (This page is not shown if IntelliJ IDEA has guessed what you are importing.)
4. On the next page of the wizard, specify Gradle project settings and global Gradle settings , click Finish.

Note You can also select File | Open and select `build.gradle` file. In this case IntelliJ IDEA will import a Gradle project even if the project was not opened or imported before.

Note that if you import a multi-module project, you can make a partial import. IntelliJ IDEA displays the Gradle Project Data to Import dialog where you can select modules that you want to include in your project.

Working with Gradle Projects

IntelliJ IDEA lets you link a Gradle project to an existing IntelliJ IDEA project.

When a project is created by importing from Gradle model , this link is established automatically and the Gradle tool window is enabled.

If you open your project with a valid `buid.gradle` file using File | Open command, the Gradle tool window will be disabled.

If an IntelliJ IDEA project is not linked to a Gradle project, then the Gradle tool window is also disabled.

In this case, IntelliJ IDEA displays a message with a link that quickly lets you import your gradle project and enable the Gradle tool window.

If the Gradle tool window is active then you have at least one Gradle project linked.

It is also possible to open, detach or refresh a linked Gradle project. This is done in the Gradle tool window .

To link a Gradle project to IntelliJ IDEA project

1. Open Gradle tool window.
2. In the Gradle tool window, click + to attach a gradle project.
3. In the file choose dialog box that opens, select the desired `build.gradle` file, and click OK. Now as there is a linked Gradle project, the Gradle tool window shows the toolbar and tree view of Gradle entities.

To open a linked Gradle project

1. In the Gradle tool window , right-click on the linked project.
2. From the context menu select .
   IntelliJ IDEA navigates to the appropriate Gradle configuration file and the related `build.gradle` file opens in the editor.

To detach a linked Gradle project

1. In the Gradle tool window , under All tasks right-click on the linked project.
2. From the context menu, select —. Alternatively, you can select the linked project and click — on the Gradle tool bar.
3. In the Import Gradle Project pop-up window, clear the check box if you do not want to delete the IntelliJ IDEA project as well.
4. Click OK.
   The Gradle project is detached from the IntelliJ IDEA project and is not synchronized any more.

To refresh a linked Gradle project

1. In the Gradle tool window , right-click on the linked project.
2. From the context menu, select .
   The IntelliJ IDEA synchronizes all linked projects.

Gradle Selective Importing

IntelliJ IDEA lets you perform a partial import for Gradle multi-module projects.

You can select the data or modules for your partial import using one of the following procedures:

- Importing a Gradle project via project wizard
- Importing a module via project structure
- Importing a module via Gradle tool window

Importing a Gradle project via project wizard

1. Perform steps that are described in the Importing Project from Gradle Model topic.
2. When Gradle Project Data to Import Dialog opens, select modules or data that you want to import and click OK.

Importing a module via project structure

1. Open the Project Structure dialog.
2. On the Module Page , click + icon and select Import Module.
3. In the dialog that opens, select a module you want to import and click OK.
4. In the Select Project Data to Import dialog, select modules or data that you want to include in your project and click OK.

Importing a module via Gradle tool window

1. In the Gradle tool window , on the toolbar, click icon.
2. In dialog that opens select modules that you want to include in your project and click OK.

Synchronizing Changes in Gradle Project and IntelliJ IDEA Project

Once a Gradle project is imported into IntelliJ IDEA, you can explore its structure, make changes both to the structure of the original Gradle project and to the resulting IntelliJ IDEA project, synchronize structures of the two projects, and explore differences.

In this section:

Synchronizing Structure of Gradle and IntelliJ IDEA Projects

- In the Gradle tool window , click refresh button .

  On pressing this button, IntelliJ IDEA parses the project structure and displays detected differences (if any) in the Gradle tool window.

  Note that if you configure a dependency through the Project Structure dialog, the dependency will only appear in the IntelliJ IDEA Project tool window, not in the Gradle tool window.

Tip Every time refresh action is performed, progress is shown in the Status bar .

Working with Gradle Tasks in Gradle Tool Window

Gradle tool window lets you create a run/debug configuration for the selected Gradle task. You can also execute and debug different Gradle tasks.

To create run/debug configuration for Gradle tasks in Gradle Tool Window

1. In the Gradle tool window , in the Tasks area, double-click on the Gradle project.

   The list of tasks opens.

2. In the list of tasks, select a task for which you want to create a run/debug configuration, right-click on it and from the context menu select Create option.

   The Create Run/Debug Configuration dialog opens.

3. In the Create Run/Debug Configuration dialog, specify the necessary information and click OK.

To debug Gradle tasks in Gradle Tool Window

Tasks available for debugging via Gradle tool window are the ones that implement `org.gradle.process.JavaForkOptions` interface. For example, test or run . For debugging Gradle script tasks themselves, use a Groovy run/debug configuration that appears in the context menu when you right-click on the script task in the editor.

1. In the Gradle tool window , in Tasks area, double-click on the Gradle project.

   The list of tasks opens.

2. In the list of tasks, select a task which you want to debug, right-click on it and from the context menu select Debug option.

   The debugging process is started and the task is added to the list of recent tasks located under the Run Configurations node.

To execute Gradle tasks in Gradle Tool Window

1. In the Gradle tool window , in Tasks area, double-click on the Gradle project.

   The list of tasks opens.

2. In the list of tasks, select the task that you want to execute and double-click on it. Alternatively, right-click on the selected task and from the context menu select Run option.

   The task is executed and is added to the list of recent tasks.

Associating Gradle Tasks with Keyboard Shortcuts

You can associate a Gradle task with a keyboard shortcut and execute commonly-used tasks with a single key-stroke. In a Gradle project, its tasks appear under the Gradle node in the Keymap dialog.

To associate a keyboard shortcut with a Gradle task

1. In the Gradle Projects tool window , right-click the desired task.
2. On the context menu, choose Assign Shortcut. Keymap dialog appears.
3. Configure keymap, as described in the section Configuring Keyboard Shortcuts .

Maven

Maven is a software tool that helps you manage Java projects, and automate application builds. IntelliJ IDEA fully integrates with Maven version 2.2 and later versions, allowing you to create or import Maven modules, download artifacts and perform the goals of the build lifecycle and plugins.

In this section:

- Maven
  - Maven support
  - Maven Module
  - Maven repositories
  - pom.xml
  - Import of Maven projects
  - Maven Projects tool window
  - Running and debugging Maven goals
  - Compiling
  - WAR overlays
  - Dependency Graph
- Getting Started with Maven
- Importing Project from Maven Model
- Creating a Maven Module
- Creating Maven Run/Debug Configuration
- Common Maven Procedures

Maven support

Maven integration is shipped with IntelliJ IDEA, and you do not need to perform any additional actions to install it. You can start using it straight away for importing Maven projects, working with them and for running Maven goals.

Maven support in IntelliJ IDEA includes the following features:

- Dedicated module type
- Maven repositories support
- Full editing support for pom.xml file
- Possibility to import Maven projects
- Running and debugging Maven goals
- Compiling
- WAR overlays
- Dependency Graph

Maven Module

For the Maven projects IntelliJ IDEA provides a dedicated module type. For each Maven Module, IntelliJ IDEA creates a `pom.xml` file. So doing, a Maven Module can be created either with the basic `pom.xml` file, or from a certain pattern called Maven archetype.

The dedicated module type allows creating Maven projects that have parent and aggregator Maven projects.

Maven repositories

IntelliJ IDEA enables communication with remote Maven repositories, and maintaining the local ones.

When Maven goals are executed in the IDE, IntelliJ IDEA is aware of all downloads and artifacts. However, if you launch Maven from command line, the artifacts produced come unnoticed, and you have to make IntelliJ IDEA search for updates. For this purpose, IntelliJ IDEA suggests a quick fix to update indices, and a node in the Settings dialog (Maven | Repositories).

In particular, updating indices helps keep the list of available Maven archetypes up to date.

pom.xml

Maven works with `pom.xml` files to build projects. At minimum, a `pom.xml` file should contain a root element `<project>`, and identifiers of the project group, artifact and version.

IntelliJ IDEA supports syntax of the `pom.xml` files. When editing `pom.xml` files, you can enjoy the following advanced editing features:

- Syntax highlighting.
- Maven dependencies and parent generation using `Alt+Insert`.
- Quick fixes for adding dependencies and updating Maven indices.
- Code completion.
- Navigation between modules and `pom.xml` dependencies (`Ctrl+B`), Navigate | Declaration on a property from `pom.xml` files, `settings.xml` and `profiles.xml` files, system and environment properties, and properties defined in custom Maven filters.
- Structure view.
- Find Usages (`Ctrl+F7`).
- Code folding.
- Reformatting.
- Rename refactoring for properties defined in Maven project and custom filters files.

– Validation.
– Viewing parameter information `Ctrl+P`.
– Viewing quick info `N/A`.

Import of Maven projects

If you want to use an existing Maven project, you can import it directly by opening its `pom.xml` file. When a Maven project is imported, it maps to an IntelliJ IDEA module with the name, which is equal to the Maven project's `artifactId`. Dependencies between the Maven projects map to the dependencies of IntelliJ IDEA modules from the libraries and other modules. IntelliJ IDEA analyzes the `pom.xml` file and automatically downloads the necessary dependencies.

Maven Projects tool window

The dedicated Maven Projects tool window allows you to manage Maven projects, configure preferences for the current Maven project and the defaults for the new ones, and execute Maven goals. Results are displayed in the Maven Build Output window.

Running and debugging Maven goals

IntelliJ IDEA provides two ways of running the Maven goals:

– Create run/debug configuration and launch it.
– Use the Run Maven Build command in the Maven Projects tool window. This way doesn't require any run/debug configuration.

Compiling

IntelliJ IDEA's Make features are capable of filtering Maven resources. However, IntelliJ IDEA yet does not support filtering web resources.

Details on configuring filter options within the `pom.xml` can be found at http://maven.apache.org/plugins/maven-resources-plugin/.

Note, that classpath for Maven-based projects is built following the Maven rules. IntelliJ IDEA supports *compile*, *test* and *run-time* dependency scopes.

WAR overlays

IntelliJ IDEA Maven integration correctly handles WAR overlays, which is important for the Web projects that use common resources defined in a WAR module. IntelliJ IDEA unpacks such WAR to the `overlays` directory under the content root of the dependent Maven Module.

Dependency Graph

IntelliJ IDEA provides you with a handy graph-view of Maven dependencies available from the context menu in the Maven Projects tool window.

Getting Started with Maven

IntelliJ IDEA lets you work with Maven by importing a project that contains `pom.xml` or by creating a Maven project from scratch.

- Creating Maven Project
- Importing Project from the Maven Model
- Working with Project Structure
- Working with POM
- Working with Maven Tool Window
- Working with Maven Settings

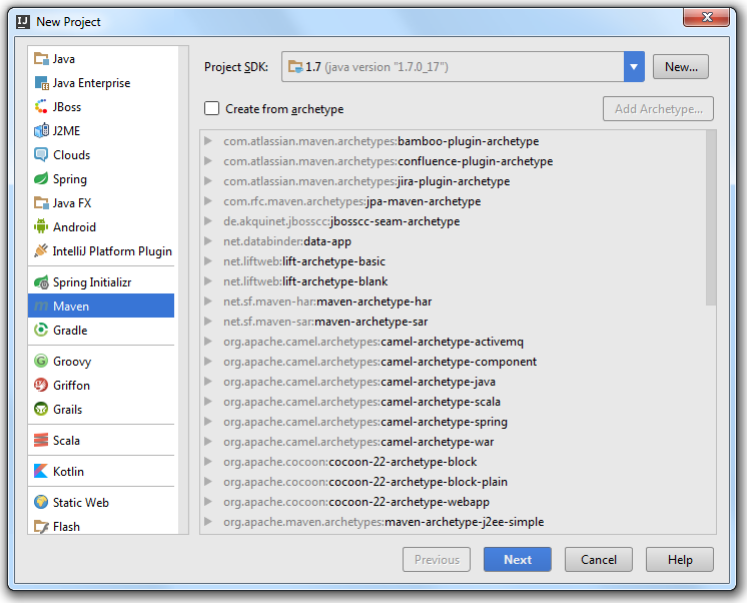
Creating Maven Project

1. Open Project Wizard, in the left-hand pane select Maven.
2. In the right-hand pane, specify the following options:
   - Project SDK - specify your project SDK (JDK).

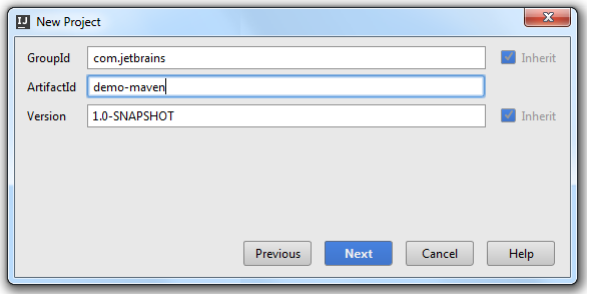
     If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select the installation folder of the desired JDK.
   - If you want to create a Maven project based on one of the Maven Archetypes, select the Create from archetype check box, and select the desired archetype from the list.

     You can click Add Archetype and add a new archetype to the list of the existing ones.

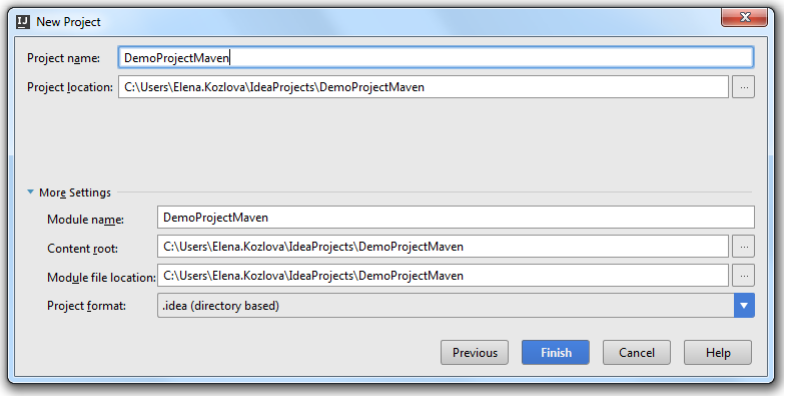
   Click Next.



3. On the next page of the wizard, specify properties for your Maven project and click Next.



4. On the next page of the wizard, specify project's information and click Finish.

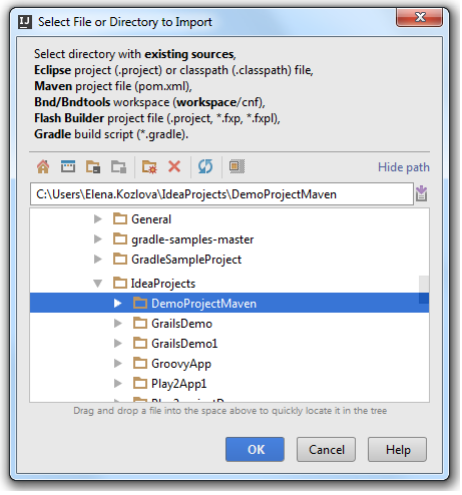


IntelliJ IDEA creates a project with `pom.xml`.

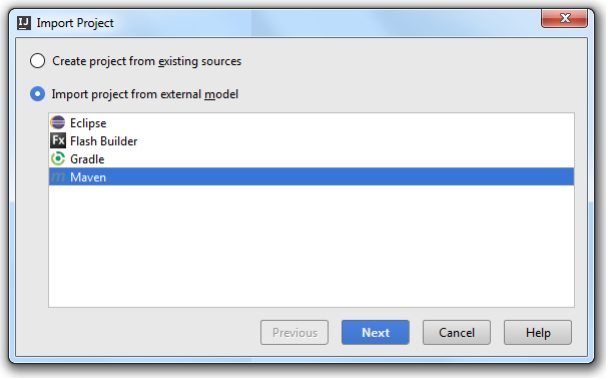
Importing Project from the Maven Model

IntelliJ IDEA enables you to create a project on the base of an existing Maven project, represented by `pom.xml`.

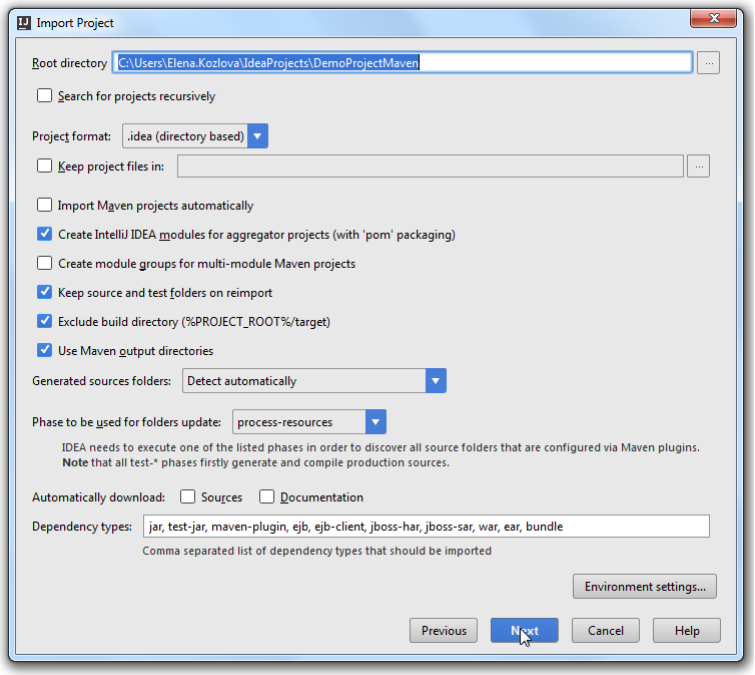
1. If no project is currently open in IntelliJ IDEA, click Import Project on the Welcome screen. Otherwise, select File | New | Project from Existing Sources.
2. In the dialog that opens, select the directory that contains the project to be imported, or a file that contains an appropriate project description. Click OK.



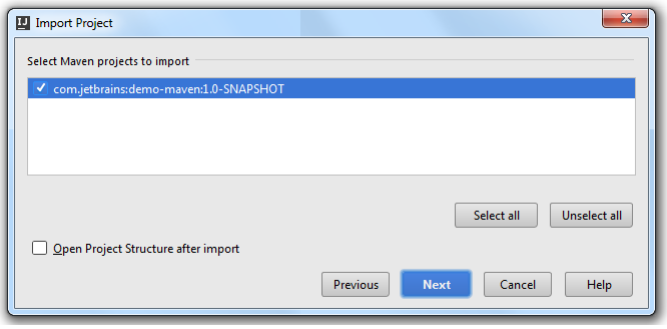
3. On the first page of the Import Project wizard, select Maven, and click Next. (This page is not shown if IntelliJ IDEA has guessed what you are importing.)



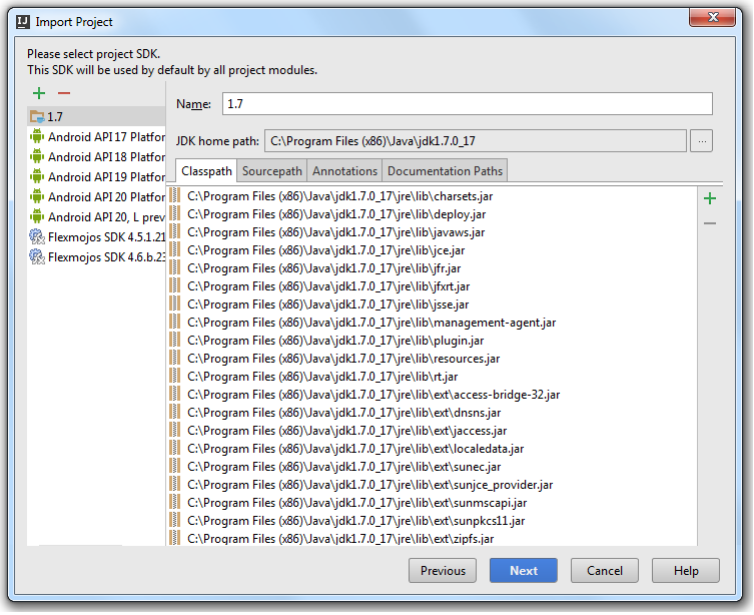
4. On the next page of the wizard, specify the project root directory. Define your import preferences, if they differ from those specified in the Maven Integration dialog. Click Environment settings to configure Maven environment settings.
   Click Next.



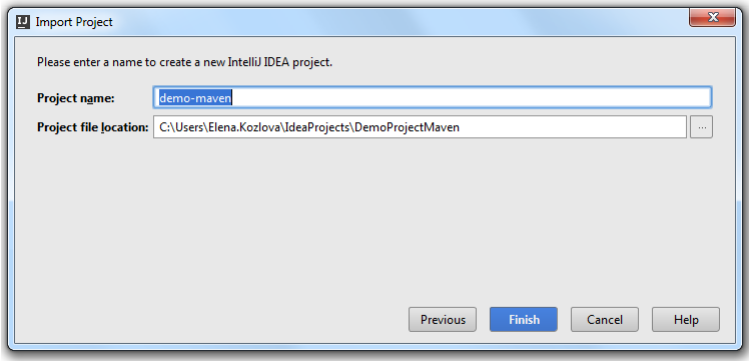
5. On the next page of the wizard, select the desired profiles, and click Next.
   Note that the Select profiles page appears if your root directory contains more than one `pom.xml` and the Search for projects recursively check box is selected.
6. On the next page of the wizard, select the projects to be imported. Note that IntelliJ IDEA detects Maven projects by scanning the specified root. Click Next.



7. On the next page of the wizard, select the project's SDK. Click Next.



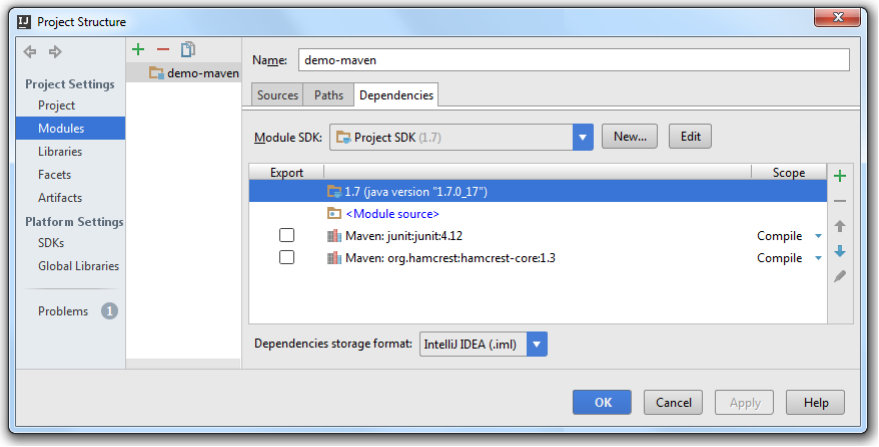
8. On the next page of the wizard, specify the new project name and location of the project files. Click Finish.



Working with Project Structure

After importing a Maven project, check its project structure to make sure that the Maven structure was imported as well.

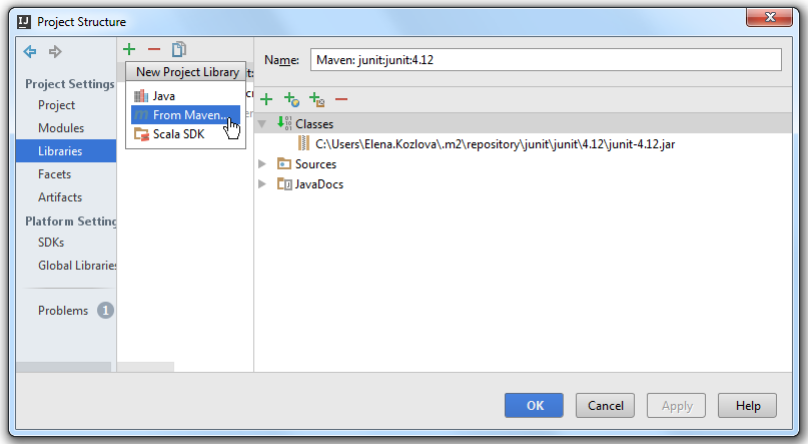
1. On the main menu select File | Project Structure.
2. On the page that opens, check that the appropriate libraries and dependencies were downloaded.



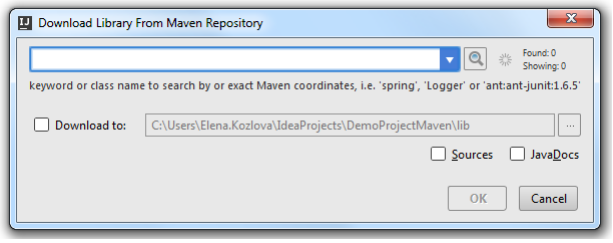
You can download an external maven library to your project using Project Structure dialog.

Open Project Structure page, from the left-hand pane, select Libraries, click + icon and from the drop-down list, select From Maven

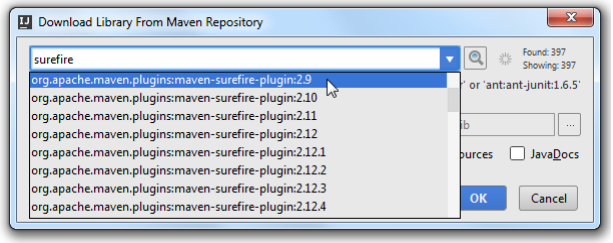
1. Open Project Structure page, from the left-hand pane, select Libraries, click + icon and from the drop-down list, select From Maven.



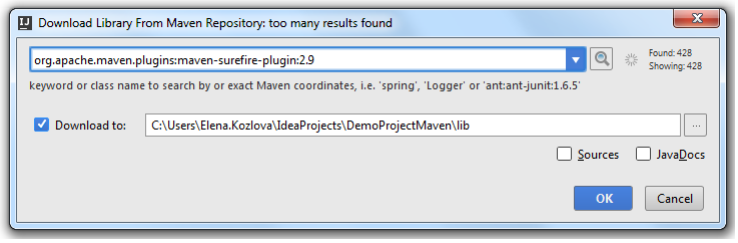
The Download Library from Maven Repository window opens.



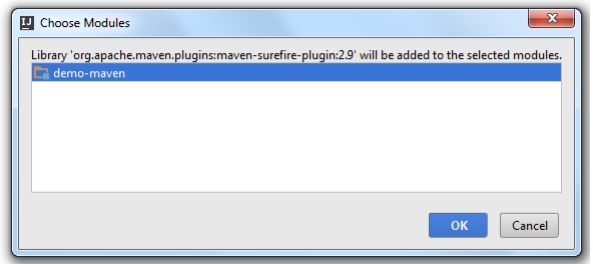
2. Start typing the name of the library in the search field and select your target from the search results' list.



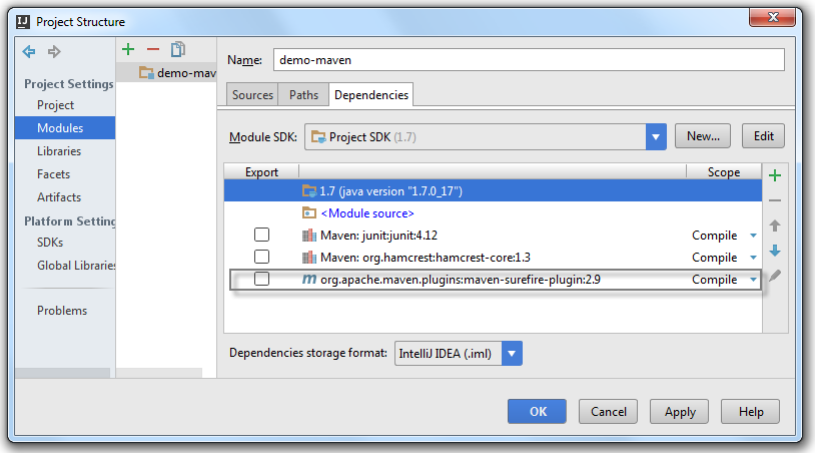
3. Select Download to check box, enter the location to which you want to download the library and press OK. By default, the Download to check box is cleared and IntelliJ IDEA downloads the library to the local repository.



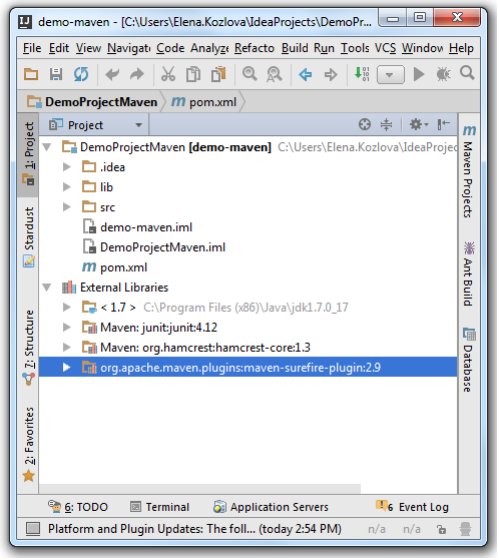
4. In the window that opens, select the module to which you want to add the maven library and press OK.



5. As a result, IntelliJ IDEA adds the library to your dependencies' list. When you open the project, IntelliJ IDEA synchronizes such libraries. It might be helpful if the libraries were deleted from your computer or the project was taken from the VCS and libraries haven't been downloaded yet.



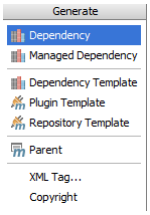
The library is also added to your project tool window.



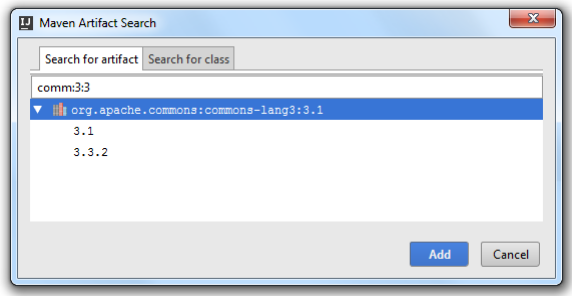
Working with POM

The code assistance is available when you work with POM in the editor. You can generate a new dependency and manage it. You can also add dependency, repository or plugin templates.

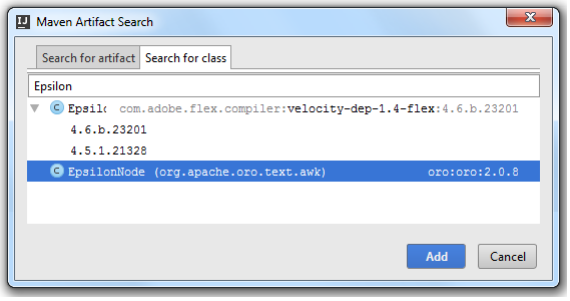
1. Open your POM file and in the editor and press Alt + Insert.
2. From the list that opens, select an item that you want to generate, for example, a dependency.



3. In the window that opens search for the artifact that you want to add and click Add. If you don't remember the full name of an artifact you can enter part of the *groupId* and part of the *artifactId* separated by a colon (:).



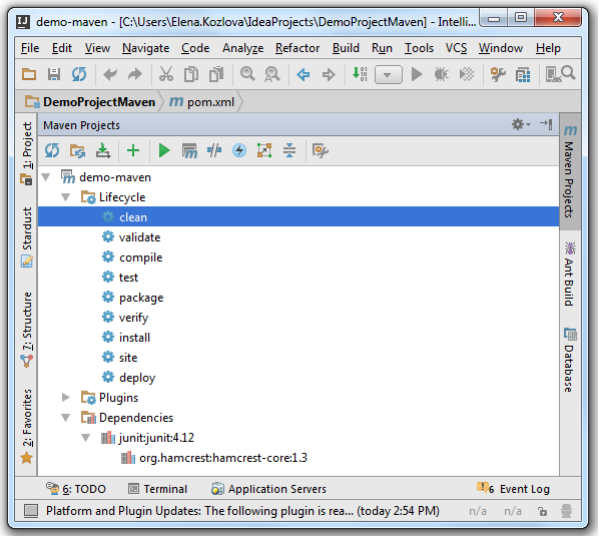
You can search either based on the name of artifact or the name of a class.



Working with Maven Tool Window

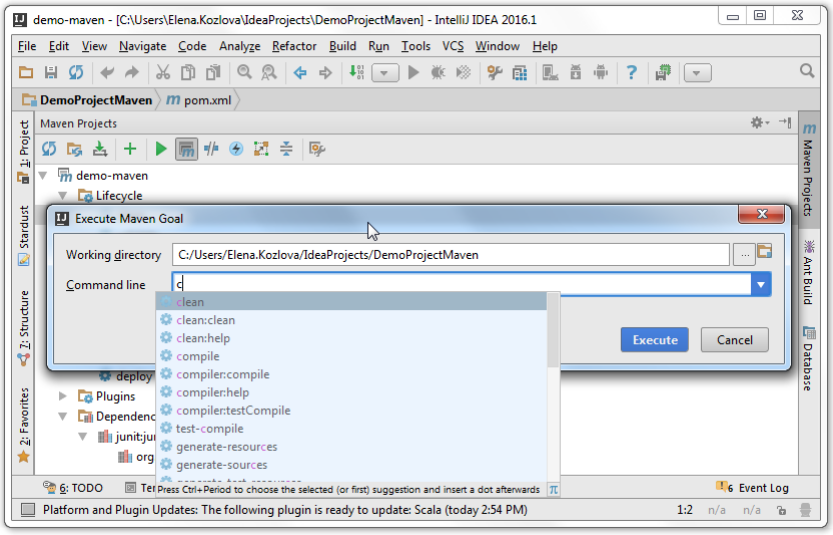
After creating a new or importing an existing Maven project, IntelliJ IDEA displays the maven model inside the Maven tool window. You can use the Maven tool window to run maven goals, create run/debug configurations, show dependencies, customize the goal execution and perform other useful tasks.

1. On the main menu click View | Tool Windows | Maven Projects to open the Maven tool window.



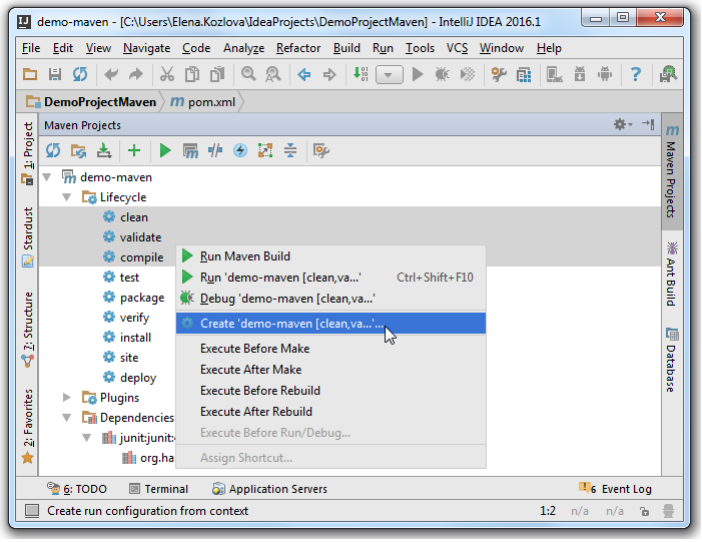
2. You can perform the following tasks:
   - run goals

     Click Execute Maven Goal icon.

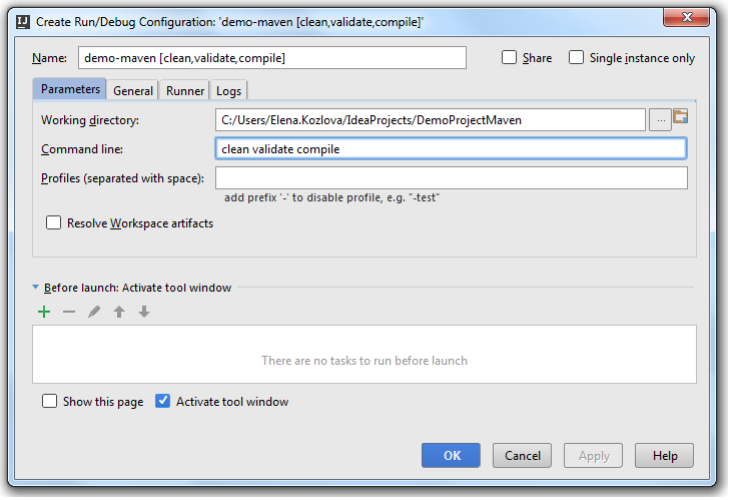


   - create run/debug configuration for the list of goals

     Highlight the goals you want to run, right-click on them and from the context menu select Create 'mvn [name of a goal]'.

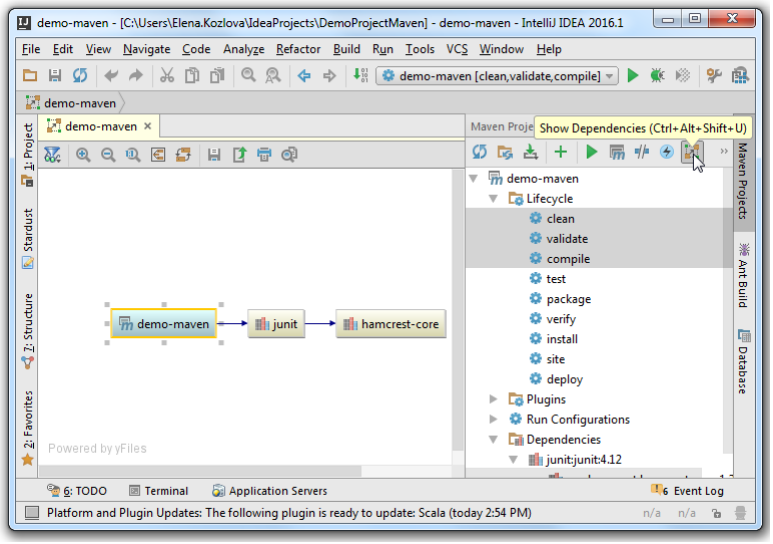


In the window that opens, specify run/debug configuration settings and click OK.



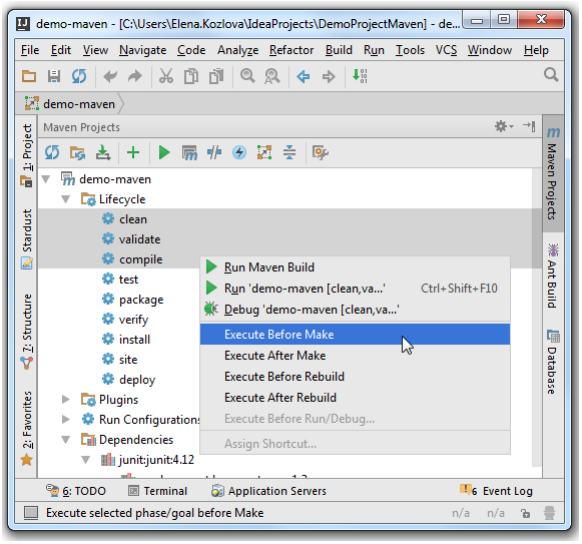
– display dependencies

Click Show Dependencies icon.

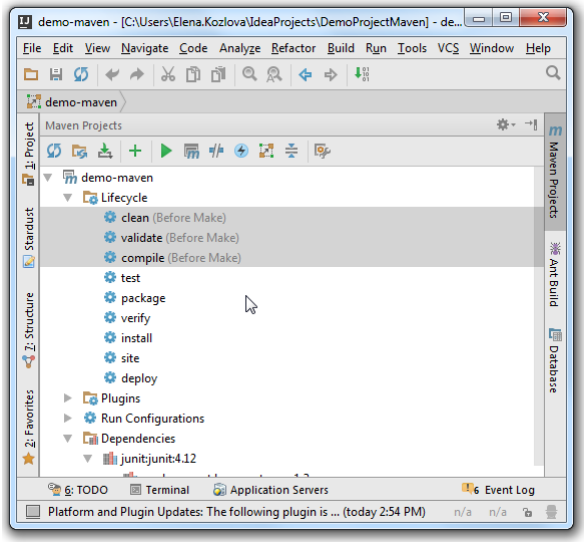


– set triggers for running maven goals

Right-click on the goal and from the context menu select at what point IntelliJ IDEA should run your goal.

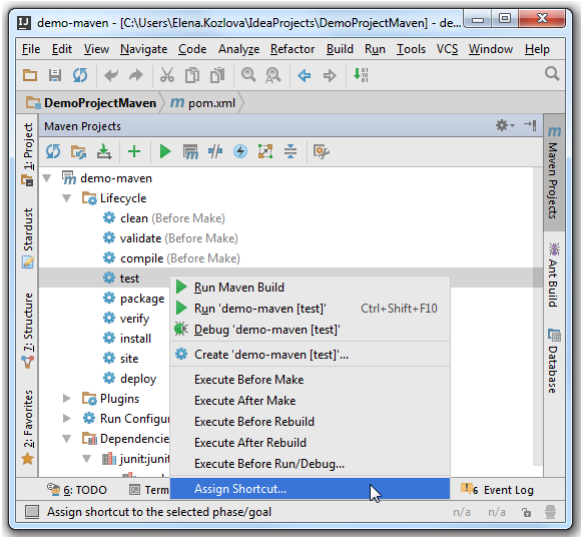


As a result, the trigger for the goal execution is displayed next to the selected goal.

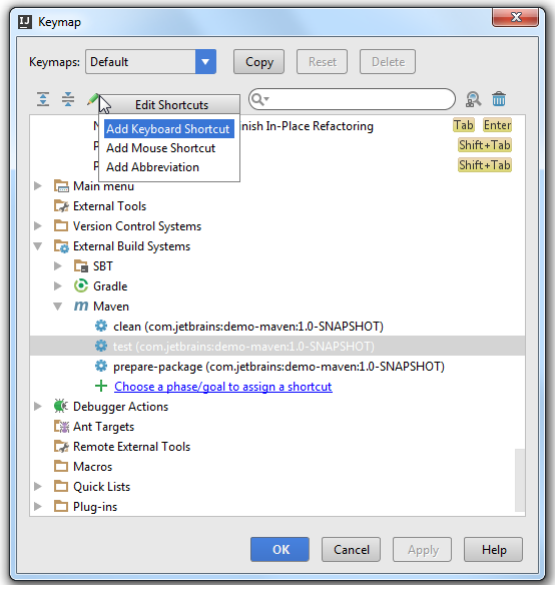


– assign a shortcut for running maven goal

Right-click on the goal and from the context menu select Assign Shortcut.

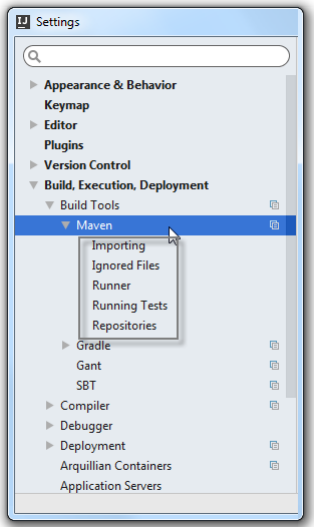


In the window that opens, add the shortcut and press OK.



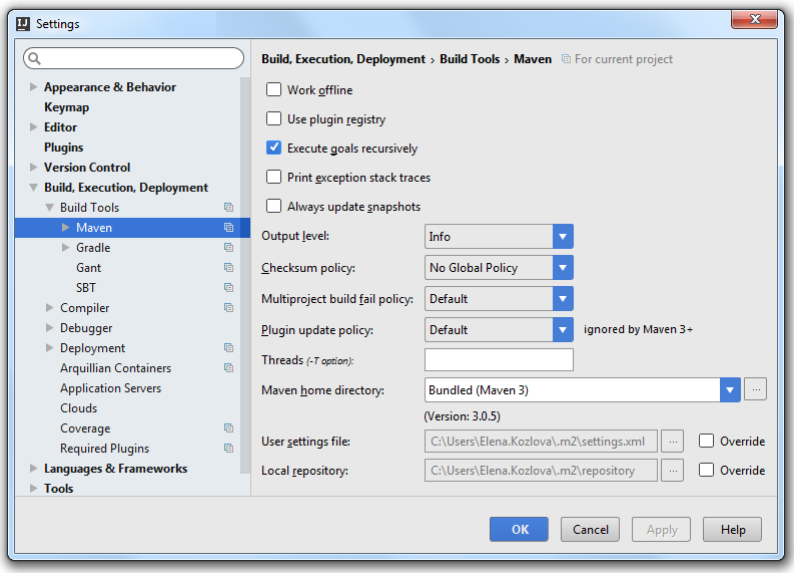
Working with Maven Settings

IntelliJ IDEA lets you change maven settings that are specific for your current project. Under Maven category you can find a number of options that can help you configure how your project is set up.



For example, you can choose to work in offline mode, add repositories or change how things are imported in your project.

1. On the main menu select File | Settings | Build, Execution, Deployment | Build Tools | Maven.



2. In the window that opens specify settings for your current maven project and click OK.

Importing Project from Maven Model

IntelliJ IDEA enables you to create a project on the base of an existing Maven project, represented by `pom.xml` file. There are two ways to import a Maven project:

- Using the New Project wizard, with the custom settings .
- Using the Open Project dialog, with the default settings .

To import a Maven project

1. If no project is currently open in IntelliJ IDEA, click Import Project on the Welcome screen . Otherwise, select File | New | Project from Existing Sources.
2. In the dialog that opens , select the directory that contains the project to be imported, or a file that contains an appropriate project description. Click OK.
3. On the first page of the Import Project wizard, select Maven, and click Next. (This page is not shown if IntelliJ IDEA has guessed what you are importing.)
4. On the next page of the wizard, specify the project root directory. Define your import preferences, if they differ from those specified in the Maven Integration dialog . Click Environment settings to configure Maven environment settings.
   Click Next.
5. On the next page of the wizard, select the desired profiles, and click Next.
6. On the next page of the wizard, select the projects to be imported. Note that IntelliJ IDEA detects Maven projects by scanning the specified root. Click Next.
7. On the next page of the wizard, select the project's SDK. Click Next.
8. On the next page of the wizard, specify the new project name and location of the project files. Click Finish.

To import a Maven project with default settings

1. On the main menu, choose File | Open.
2. In the dialog box that opens, select the desired `pom.xml` file, and click OK. In this case the import is performed automatically, with the settings defined in the Maven Integration dialog .

Creating a Maven Module

Maven module enables managing Maven projects, executing Maven goals, downloading artifacts, etc.

When a Maven project is created, its contents is shown in the Maven Projects tool window, and the powerful editing assistance is available for the `pom.xml` files.

To create a Maven module from scratch

1. Open the project you want to add a module to, and select File | New | Module.
   As a result, the New Module wizard opens. (There are also other ways of accessing this wizard, see Opening the New Module wizard.)

2. On the first page of the wizard , in the left-hand pane, select Maven .

3. In the right-hand part of the page:
   - Specify the SDK (JDK) to be used.
     If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select the installation folder of the desired JDK.
   - If you want to create a Maven module based on one of the Maven Archetypes, select the Create from archetype check box, and select the desired archetype from the list.

   Click Next.

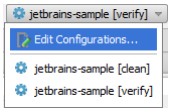
4. On the Maven page of the wizard , specify the aggregator and parent Maven projects, if any, and Maven project coordinates. Note that if a parent Maven project is specified, the new Maven project can inherit its GroupId and ArtifactId.
   Click Next.

5. Specify the name and location settings for your module. For more information, see Module Name and Location .
   Click Finish.

Creating Maven Run/Debug Configuration

IntelliJ IDEA provides Maven-specific form of run/debug configuration. When a Maven project is imported, each goal gets its own configuration, with the pre-defined name, project file, profiles, goals, and other settings.

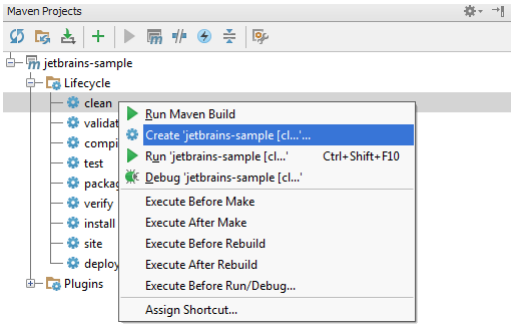
You can save such configuration for future reuse, or create your own ones. When a configuration is created, it is added to the list of permanent configurations and becomes available from the drop-down list on the main toolbar:



In general, working with run/debug configurations follows the same procedure as described in the section Creating and Editing Run/Debug Configurations . This section provides a *shortcut* procedure, specially tailored for Maven integration.

To create a run/debug configuration for a Maven goal

1. In the Maven Projects tool window, expand a project and right-click the desired goal.
2. On the context menu of the selection, choose Create '<project>[goal]' :



   The Run/Debug Configuration dialog box for Maven opens, with the current Maven project, goal, profiles and other settings.

3. Edit the run/debug configuration as required, and click OK.

Common Maven Procedures

This section covers common Maven procedures:

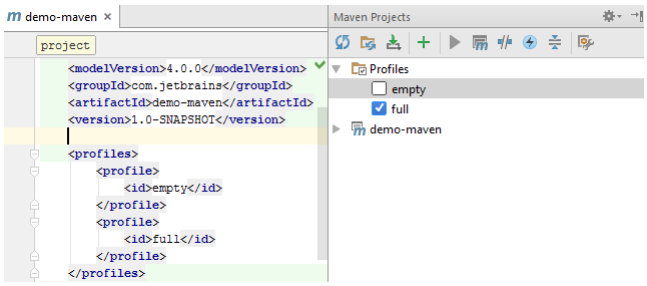
- Activating and Deactivating Maven Profiles
- Associating Maven Goals with Keyboard Shortcuts
- Configuring Triggers for Maven Goals
- Downloading Libraries from Maven Repositories
- Executing Maven Goal
- Editing Maven Settings
- Generating Maven Dependencies
- Resolving References with Maven
- Working with Maven Dependencies
- Working in Offline Mode

Activating and Deactivating Maven Profiles

Build profiles define the type of executing Maven goals, and affect the way Maven projects are imported. For example, you can define dedicated release, test or debug profiles.

Profiles are declared in the `<profiles>` section of a `pom.xml` file. You can view the profiles defined in your `pom.xml` files in the Profiles list of the Maven Projects tool window.

Adding `profile` tag to the `profiles` section of a `pom.xml` file adds a new profile check box under the Profiles node:



Warning! Do not forget to sync every time you change your `pom.xml` file in order to view the changes in the Maven Projects tool window. When the `pom.xml` is changed, a pop-up window is displayed suggesting you to synchronize it. You can do it using the Re-import button in the editor, or button in the toolbar of the Maven Projects tool window.

To toggle between active and inactive state of a profile

1. In the Maven Projects tool window , expand the Profiles node and select the desired profile.
2. To activate a profile, select its check box. To deactivate profile, clear its check box.

Associating Maven Goals with Keyboard Shortcuts

You can associate a Maven goal with a keyboard shortcut and execute commonly-used goals with a single key-stroke. In a Maven project, its goals appear under the Maven goals node in the Keymap dialog.

To associate a keyboard shortcut with a Maven goal

1. In the Maven Projects tool window , right-click the desired goal.
2. On the context menu, choose Assign Shortcut. Keymap dialog appears.
3. Configure keymap, as described in the section Configuring Keyboard Shortcuts .

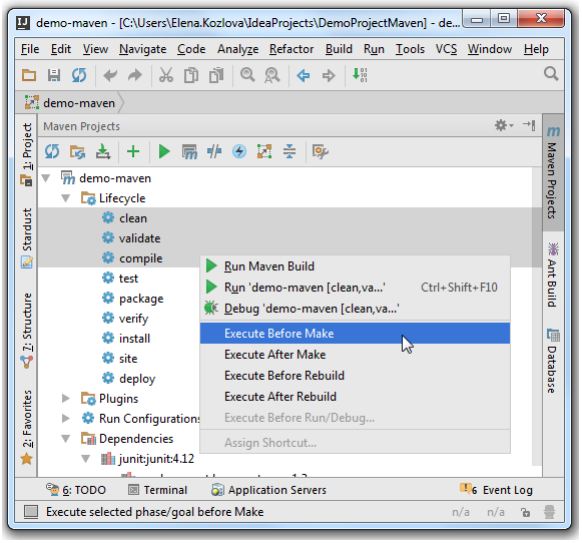
Configuring Triggers for Maven Goals

You can configure triggers that execute Maven goals before or after certain events, for example, compiling, running, or testing. These triggers are called *compilation triggers* and *execution triggers* respectively.

To set up a trigger

1. On the Maven Projects tool window , right-click the desired node (a phase of the lifecycle, or a plugin).

2. On the context menu, click one of the options:
   - Execute Before Make
   - Execute After Make
   - Execute Before Rebuild
   - Execute After Rebuild
   - Execute Before Run/Debug



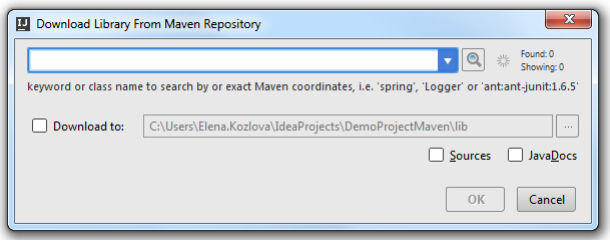
Downloading Libraries from Maven Repositories

IntelliJ IDEA makes it possible to find and download libraries from the public Maven repositories, bypassing the `pom.xml` file. This way, you can keep abreast with ever-changing external libraries stored in public Maven repositories, without creating a Maven project. So doing, one can use the Maven coordinates to uniquely identify the required libraries.

To download libraries from the public Maven repositories

1. Open the Project Structure dialog.
2. Under Project Settings, click Libraries.
3. In the central part of the Libraries page, click + to add a new library.
4. In the New Project Library window, select From Maven to add a new library from the Maven repository.
5. In the dialog box that opens, specify the following parameters:
   - The desired Maven coordinates, class name or a keyword.
   - The path to the target directory.

   You can also opt to download sources and documentation, by selecting the corresponding check boxes.



6. Click OK. IntelliJ IDEA downloads the library and stores it in the specified location. In the right part of the Project Settings page, you can see the details of the added library and perform the following actions:
   - Edit library.
   - + add or – detach library files.

Tip You can add your company repository to the list of searched repositories, if it is managed by Nexus. To do that, provide the Nexus URL in the section Artifactory or Nexus service URLs in the Repository Indices node of the Settings dialog.

Executing Maven Goal

There are two ways to execute a Maven goal:

- Use Run Goal command , which produces the same result as launching `mvn` from the command line.
- Run the Maven goal as an application with a Maven-specific run/debug configuration, which can be stored and reused for debugging Maven plugins.

Results are displayed in the console of the Run tool window .

To build a lifecycle phase or a plugin

1. Select the desired node in the Maven Projects tool window.
2. Choose Run Goal on the context menu of the selection.

To run a Maven goal as an application, do one of the following

- Perform the regular procedure, as described in the section Running Applications .
- Perform the *shortcut* procedure, which uses a Maven-specific run/debug configuration for each goal:
  Select the desired node in the Maven Projects tool window, and choose Run <goal>[run/debug configuration] on the context menu, or press `Ctrl+Shift+F10` .

Editing Maven Settings

A project is imported or created according the default Maven integration settings. Further, you can configure the current project in the Maven Integration dialog invoked by the button. These settings are used on project reopening and synchronizing.

Besides settings that apply to the current project, IntelliJ IDEA provides the possibility to configure the default settings, which apply to all newly created or imported projects. These settings are available in the Maven Integration: Default settings dialog, invoked by the button, or in the Template Project settings .

To configure the Maven Integration settings

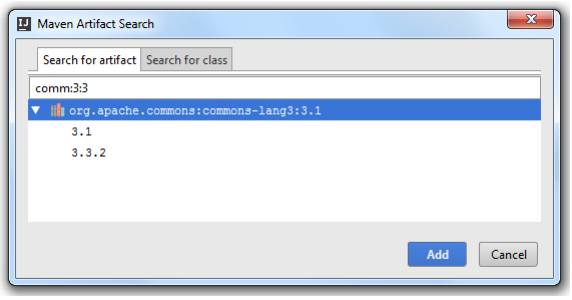
1. On the toolbar of the Maven Projects tool window, click button for the current project, or button for the default settings.
2. In the Maven page of the Settings dialog, specify the desired options. Refer to the dialog description for details.
3. Apply changes and close the dialog.

Generating Maven Dependencies

Maven integration facilitates working with dependencies. IntelliJ IDEA helps generate dependencies in the `pom.xml` files.

To create Maven dependencies in a pom.XML file

1. Open the desired `pom.xml` file for editing.
2. With the editor tab having the focus, choose Code | Generate on the main menu, or press `Alt+Insert`, and choose Dependency from the Generate pop-up window.
3. In the Maven Artifact Search dialog box, select the desired dependency to be added. Note that you can search either in the artifacts, or in the classes. To do that, type the search string in the text field to narrow down the search scope, select the desired dependency, and click Add.

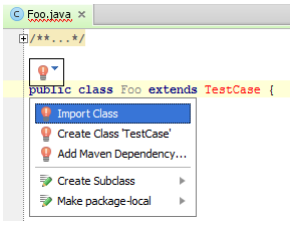


The dependency is added to your `pom.xml`. IntelliJ IDEA also adds a Dependencies node with your generated dependency to the Maven Projects tool window.

Note When searching in artifacts, the search string can refer to the ArtifactId, GroupId, and version of an artifact.
– When searching in classes, IntelliJ IDEA searches through all the available artifacts, and adds all libraries, where the class with the specified name is detected.

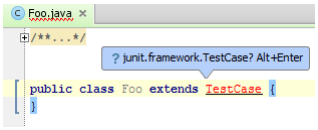
To fix unresolved reference in a Java class

1. In a Java class, place the caret at an unresolved reference, and press `Alt+Enter`.
2. From the suggestion list, choose Add Maven Dependency



The Maven Artifact Search dialog box opens with the Search for class tab having the focus, and with the reference in question as the search string.

3. Select the desired dependency, and click Add. IntelliJ IDEA downloads the necessary libraries and rebuilds indices, and adds the corresponding `<dependency>` element to the `pom.xml` file. Import assistant pop-up window appears at the unresolved reference:



4. Press `Alt+Enter` to apply import assistance.

Working with Maven Dependencies

IntelliJ IDEA makes it possible to view Maven dependencies as a UML class diagram. In this diagram, you can discern:

- Your own modules (blue)
- Test dependencies (green)
- Conflicting or unsatisfied dependencies (red)

In the Maven Dependencies diagram, you can:

- Navigate from a node to the corresponding source code.
- Mark dependencies as excluded.
- Filter out the nodes on diagram, when the diagram is too large.

To view dependencies in a UML class diagram

- Do one of the following:
  - In the Maven Projects tool window , right-click the desired `pom.xml` file, and choose Show Dependencies, or Show Dependencies Popup.
  - Right-click `pom.xml` in the editor, and choose Dependencies | Show Dependencies /Show Dependencies popup.
  - Press `Ctrl+Shift+Alt+U` or `Ctrl+Alt+U`.

To navigate from a node in the diagram to its source code

- Select the desired node, and press `F4`, or choose Jump to Source on its context menu. The corresponding file opens in the editor.

To exclude a dependency

1. Select a dependency in diagram.
2. On the context menu, choose Exclude.
3. From the drop-down list, select the module where exclusion definition will be added. The selected dependencies will be removed from diagram, and `exclusion` section will be added to the corresponding dependency.

Tip You can undo this operation by pressing `Ctrl+Z`.

To filter out the elements that are of now interest

1. On the diagram toolbar, click .
2. From the drop-down list, select the desired scope.

Working in Offline Mode

On each launch, Maven visits the remote repositories and checks for updates. Executing a Maven command can result in downloading new archives and changing Maven itself. When you switch to offline mode, Maven has to use those resources that are available locally, and report about the problems, if something is missing.

The offline mode is helpful, when you need to work offline, or when your network connection is slow.

To toggle offline mode, do one of the following

- On the toolbar of the Maven Projects tool window, click button.
- Open Maven Integration dialog , and in the General tab check the Work Offline option.

CoffeeScript Support

In this section:

- CoffeeScript Support
  - CoffeeScript support overview
  - Preparing for CoffeeScript development
  - Coding Assistance
- Transpiling CoffeeScript to JavaScript
- Running CoffeeScript
- Debugging CoffeeScript

CoffeeScript support overview

IntelliJ IDEA provides CoffeeScript support. IntelliJ IDEA recognizes `*.coffee` files, and allows you to edit them providing full range of coding assistance without any additional steps from your side. CoffeeScript files are marked with icon.

To run, debug, and test your code, you will need it translated into JavaScript which requires a transpiler and Node.js. For more details on CoffeeScript transpilation, see Transpiling CoffeeScript to JavaScript .

Preparing for CoffeeScript development

To prepare for CoffeeScript development, follow these steps

1. Make sure the CoffeeScript and Node.js plugins are installed and enabled.
   These plugins are not bundled with IntelliJ IDEA, but they are available from the JetBrains plugin repository . Once enabled, the plugins are available at the IDE level, that is, you can use them in all your IntelliJ IDEA projects. See Enabling and Disabling Plugins for details.
2. Download and install the Node.js framework.
3. Configure the Node.js interpreter in IntelliJ IDEA:
   2. On the Node.js and NPM page that opens, specify the location of the desired Node.js interpreter.

Coding Assistance

CoffeeScript support includes:

- Code completion for keywords, labels, variables, parameters and functions.
- Error and syntax highlighting.
- Code formatting and folding .
- Refactoring:
  - Common refactoring procedures, such as extract method , inline, rename/move, etc.
  - CoffeeScript-specific refactoring procedures, such as change signature, extract parameter, extract variable.

  See JavaScript-Specific Refactorings for details.
- Code generation
  - Generating code stubs based on file templates during file creation.
  - Ability to create line and block comments (`Ctrl+Slash` / `Ctrl+Shift+Slash`).
- Navigation through source code
  - Navigating with Structure View .
  - Navigate | Declaration (`Ctrl+B`).
  - Navigate | Implementation (`Ctrl+Alt+B`) from overridden method / subclassed class.
  - Navigate | Symbol (`Ctrl+Shift+Alt+N`).
- Compiling to JavaScript for further running, debugging, and testing, see Running CoffeeScript and Debugging CoffeeScript .
- Executing CoffeeScript files involves:
  - Ability to preview results of CoffeeScript files compilation to JavaScript.
  - Ability to launch CoffeeScript files from the context menu.
  - Run/debug configuration for NodeJS includes the ability to use CoffeeScript plugin.

Transpiling CoffeeScript to JavaScript

On this page:

- Introduction
- Before you start
- Installing the CoffeeScript transpiler globally
- Installing the CoffeeScript transpiler in a project
- Creating a File Watcher
- Examples of customizing the behaviour of a transpiler
- Transpiling the CoffeeScript code
- Previewing the transpilation results without running a transpiler

Introduction

CoffeeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, CoffeeScript code has to be translated into JavaScript. This operation is referred to as transpilation and the tools that perform it are called transpilers.

IntelliJ IDEA supports integration with the coffee-script transpilation tool. The tool translates CoffeeScript code into JavaScript and creates source maps that set correspondence between lines in your CoffeeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.

In IntelliJ IDEA, transpiler configurations are called File Watchers. For each supported transpiler, IntelliJ IDEA provides a predefined File Watcher template. Predefined File Watcher templates are available at the IntelliJ IDEA level. To run a transpiler against your project files, you need to create a project-specific File Watcher based on the relevant template, at least, specify the path to the transpiler to use on your machine.

The easiest way to install the CoffeeScript transpiler is to use the Node Package Manager (npm), which is a part of Node.js . See Installing and Removing External Software Using Node Package Manager for details.

Depending on the desired location of the CoffeeScript transpiler executable file, choose one of the following methods:

- Install the transpiler globally at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project.
- Install the transpiler in a specific project and thus restrict its use to this project.
- Install the transpiler in a project as a development dependency .

In either installation mode, make sure that the parent folder of the CoffeeScript transpiler is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

IntelliJ IDEA provides user interface both for global and project installation as well as supports installation through the command line.

Before you start

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The CoffeeScript transpiler is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the CoffeeScript transpiler.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the CoffeeScript transpiler and npm from any folder.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Install and enable the File Watchers repository plugin.
   The plugin is not bundled with IntelliJ IDEA, but it is available from the IntelliJ IDEA plugin repository plugin repository . See Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins for details.

Installing the CoffeeScript transpiler globally

Global installation makes a transpiler available at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project. Moreover, during installation the parent folder of the transpiler is automatically added to the `PATH` variable, which enables you to launch the transpiler from any folder. To install the transpiler globally, do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
  3. Type the following command at the command line prompt:

     ```
     npm install -g coffee-script
     ```

     The `-g` key makes the transpiler run in the global mode. Because the installation is performed through NPM, the CoffeeScript transpiler is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

     For more details on the NPM operation modes, see npm documentation . For more information about installing the CoffeeScript transpiler, see https://npmjs.org/package/coffee-script .
- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click .
  3. In the Available Packages dialog box that opens, select the required package to install.
  4. Select the Options check box and type `-g` in the text box next to it.

5. Optionally specify the product version and click Install Package to start installation.

Installing the CoffeeScript transpiler in a project

Installing a transpiler in a specific project restricts its use to this project. To run project installation, do one of the following:

– Run the installation from the command line:

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install coffee-script
```

– Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in template-based projects of the type Node Boilerplate or Node.js Express, which already have the `node_modules` folder. The latter is important because NPM installs the CoffeeScript transpiler in a `node_modules` folder. If your project already contains such folder, the CoffeeScript transpiler is installed there.

Projects of other types or empty projects may not have a `node_modules` folder. In this case npm goes upwards in the folder tree and installs the CoffeeScript transpiler in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be outside your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the CoffeeScript transpiler is installed there.

In either case, make sure that the parent folder of the CoffeeScript transpiler is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

Creating a File Watcher

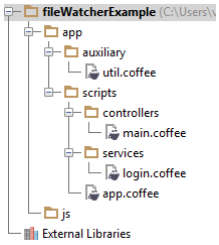
IntelliJ IDEA provides a common procedure and user interface for creating File Watchers of all types. The only difference is in the predefined templates you choose in each case.

1. To start creating a File Watcher, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X on the main menu, and then click File Watchers under the Tools node. The File Watchers page that opens, shows the list of File Watchers that are already configured in the project.
2. Click the Add button + or press `Alt+Insert` and choose the CoffeeScript predefined template from the pop-up list. Your code will be translated to JavaScript and supplied with generated source maps .
3. In the Program text box, specify the path to the `coffee.cmd` file. Type the path manually or click the Browse button ... and choose the file location in the dialog box that opens.
4. Proceed as described on page Using File Watchers .

Examples of customizing the behaviour of a transpiler

Any transpiler is an external, third-party tool. Therefore the only way to influence a transpiler is pass arguments to it just as if you were working in the command line mode. Below are two examples of customizing the default output location for the CoffeeScript transpiler.

Suppose, you have a project with the following folder structure:



By default, the generated files will be stored in the folder where the original file is. You can change this default location and have the generated files stored in the `js` folder. Moreover, you can have them stored in a flat list or arranged in the folder structure that repeats the original structure under the `app` node.

– To have all the generated files stored in the output `js` folder without retaining the original folder structure under the `app` folder:

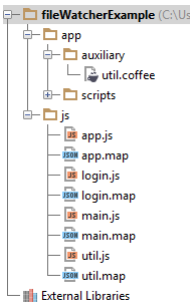
1. In the Arguments text box, type:

```
--output $ProjectFileDir$\js\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir$\js\$FileNameWithoutExtension$.js:$ProjectFileDir$\js\$FileNameWithoutExtension$.map
```

As a result, the project tree looks as follows:



– To have the original folder structure under the `app` node retained in the output `js` folder:

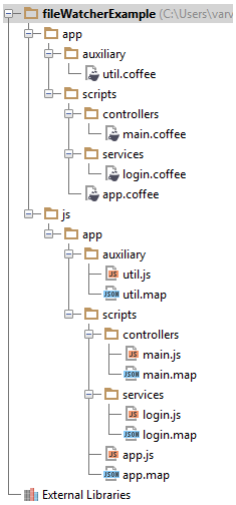
1. In the Arguments text box, type:

```
--output $ProjectFileDir$\js\$FileDirRelativeToProjectRoot$\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir$\js\$FileDirRelativeToProjectRoot$\$FileNameWithoutExtension$.js:$ProjectFileDir$\js\$FileDirRelativeToI
```

As a result, the project tree looks as follows:



Transpiling the CoffeeScript code

When you open a CoffeeScript file, IntelliJ IDEA checks whether an applicable file watcher is available in the current project. If such file watcher is configured but disabled, IntelliJ IDEA displays a pop-up window that informs you about the configured file watcher and suggests to enable it.

If an applicable file watcher is configured and enabled in the current project, IntelliJ IDEA starts it automatically upon the event specified in the New Watcher dialog .

– If the Immediate file synchronization check box is selected, the File Watcher is invoked as soon as any changes are made to the source code.

– If the Immediate file synchronization check box is cleared, the File Watcher is started upon save (File | Save All, `Ctrl+S`) or when you move focus from IntelliJ IDEA (upon frame deactivation).

The transpiler stores the generated output in a separate file. The file has the name of the source CoffeeScript file and the extension `js` or `js.map` depending on the transpiler type. The location of the generated files is defined in the Output paths to refresh text box of the New Watcher dialog . Based on this setting, IntelliJ IDEA detects the transpiler output. However, in the Project Tree, they are shown under the source `.coffee` file which is now displayed as a node.

Previewing the transpilation results without running a transpiler

IntelliJ IDEA can perform static analyses of your CoffeeScript code without actually running a transpiler and display the predicted transpilation output in the dedicated read-only viewer.

1. Open the desired CoffeeScript file in the editor, and right-click the editor background.
2. On the context menu, choose Preview Compiled CoffeeScript File. The preview is opened in the dedicated read-only viewer: the left-hand pane shows the original CoffeeScript source code and the right-hand pane shows the JavaScript code that will be generated by the transpiler when it runs.

Running CoffeeScript

On this page:

– Introduction
– Transpiling CoffeeScript manually and running the generated JavaScript code
– Transpile CoffeeScript on the fly during run

Introduction

CoffeeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, CoffeeScript code has to be translated into JavaScript. This operation is referred to as transpilation and the tools that perform it are called transpilers.

For more details about transpilation in IntelliJ IDEA, see the section Using File Watchers .

In either case, running CoffeeScript is supported only in the local mode. This means that IntelliJ IDEA itself starts the Node.js engine and the target application according to a run configuration and gets full control over the session.

For more details about running Node.js applications, see Running and Debugging Node.js .

There are two approaches to running CoffeeScript in IntelliJ IDEA:

– Transpile the CoffeeScript code manually and then run the output JavaScript code as if it were a Node.js application.
– Run the original CoffeeScript code through the NodeJS run configuration and have IntelliJ IDEA transpile it on the fly.

Transpiling CoffeeScript manually and running the generated JavaScript code

1. Transpile the CoffeeScript code into Javascript .
2. Start creating a Node.js run configuration with the following mandatory settings:
   1. The Node.js engine to use. By default, the field shows the path to the interpreter specified on the Node.js page during Node.js configuration.
   2. In the Working directory field, specify the location of the files referenced from the starting CoffeeScript file to run, for example, includes. If this file does not reference any other files, just leave the field empty.
   3. In the Path to Node App JS File text box, specify the full path to the JavaScript file that was generated from the original CoffeeScript file during the transpilation.
3. Save the configuration and click ▶ on the toolbar.
4. Proceed as while running a Node.js application .

Transpile CoffeeScript on the fly during run

1. This mode requires that the `register.js` file, which is a part of the `coffee-script` package, should be located under the project root. Therefore you need to install the `coffee-script` package on the Node.js page locally, as described in Installing and Removing External Software Using Node Package Manager .
2. Start creating a Node.js run configuration as described in Running and Debugging Node.js and specify the following mandatory settings:
   1. The Node interpreter to use. Select the relevant interpreter configuration or create a new one, see By default, the field shows the path to the interpreter specified on the Node.js page during Node.js configuration.
      For Linux and OS X, this setting is overridden by the Node.js from the path to the CoffeeScript transpiler executable file.
   2. In the Node parameters text box, type the following:

      ```
      --require coffee-script/register
      ```

   3. In the Working directory field, specify the location of the files referenced from the starting CoffeeScript file to run, for example, includes. If this file does not reference any other files, just leave the field empty.
   4. In the JavaScript file text box, specify the full path to the CoffeeScript file to run.
3. Save the configuration and click ▶ on the toolbar.
4. Proceed as while running a Node.js application .

Debugging CoffeeScript

On this page:

– Introduction

– Debugging CoffeeScript

Introduction

CoffeeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, CoffeeScript code has to be translated into JavaScript. This operation is referred to as transpilation and the tools that perform it are called transpilers.

To debug CoffeeScript in IntelliJ IDEA, you need source maps generated in addition to the JavaScript code. Source maps set correspondence between lines in your CoffeeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly. JavaScript and source maps are generated by transpiling the CoffeeScript code manually using the File Watcher of the type CoffeeScript. After that you can debug the output JavaScript code as if it were a Node.js application.

For more details about transpilation in IntelliJ IDEA, see the section Using File Watchers .

Debugging CoffeeScript is supported only in the local mode. This means that IntelliJ IDEA itself starts the Node.js engine and the target application according to a run configuration and gets full control over the session.

For more details about debugging Node.js applications, see Running and Debugging Node.js .

Debugging CoffeeScript

To transpile CoffeeScript manually and debug the output as a Node.js application

1. Set the breakpoints in the CoffeeScript code, where necessary.
2. Transpile the CoffeeScript code into Javascript using the File Watcher of the type CoffeeScript Source Map.
3. Start creating a Node.js run configuration with the following mandatory settings:
   1. The Node.js engine to use. By default, the field shows the path to the interpreter specified on the Node.js page during Node.js configuration.
   2. In the Working directory field, specify the location of the files referenced from the starting CoffeeScript file to run, for example, includes. If this file does not reference any other files, just leave the field empty.
   3. In the Path to Node App JS File text box, specify the full path to the JavaScript file that was generated from the original CoffeeScript file during the transpilation.
4. Save the configuration and click on the toolbar.
5. Proceed as while debugging a Node.js application locally .

Testing CoffeeScript

To run unit CoffeeScript unit test, you need to use dedicated packages that

ColdFusion

IntelliJ IDEA provides extensive editing support for ColdFusion files, and facilities for deploying applications to the ColdFusion server.

On this page:

- Prerequisites
- ColdFusion Support
- Configuring deployment to ColdFusion server

Prerequisites

Before you start working with ColdFusion, make sure that the CFML Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

ColdFusion server should be downloaded and installed on your machine.

ColdFusion Support

ColdFusion files are marked with icon.

ColdFusion support includes:

1. Coding assistance:
   - Code completion for tags, attributes, attribute values, functions and variable of the current scope, function arguments, functions of Java classes and components created through 'createObject' function, for components' and Java classes' names in 'createObject', for inherited methods of a component.
   - Error and syntax highlighting.
   - Code formatting and folding .
2. Numerous ways to navigate through the source code, among them:
   - Ability to open a file or component with the specified name .
   - Navigating with Structure View .
   - Navigate | Declaration (`Ctrl+B`).
3. Advanced facilities to search through the source code .
4. Code generation
   - Generating code stubs based on file templates during file creation.
   - Inserting, expanding, and generating code blocks using live templates .
   - Creating various applications elements via intention actions .
   - Ability to create line and block comments (`Ctrl+Slash` / `Ctrl+Shift+Slash`).
5. Viewing method parameters information.
6. Run/debug configuration for ColdFusion.

To configure deployment to the ColdFusion server

1. Open the Settings dialog box (`Ctrl+Alt+S`) and click the ColdFusion node.
2. On the ColdFusion page that opens, specify mappings between local folders with the application sources and the paths on the server.
   - To add a new mapping, click (`Alt+Insert`) and specify the local folder with the application sources in the Directory Path field. Type the path manually or click and choose the folder in the dialog box that opens.
     In the Logical Path field, type the URL address on the server to deploy the contents of the specified local folder to.
   - To remove a mapping from the list, select the mapping and click (`Alt+Delete`).

Context and Dependency Injection (CDI)

The Context and Dependency Injection (CDI) support in IntelliJ IDEA is based on the Java EE: Context and Dependency Injection plugin . This plugin is bundled with the IDE and enabled by default.

You can enable CDI support when creating a project or module . You can as well add CDI support for an existing module.

CDI in a module is represented by the corresponding CDI library in the module dependencies . You can download all the necessary library files right from within the IDE. You can also change the CDI version being used if and when needed.

- Making sure that the CDI Support plugin is enabled
- Enabling CDI support when creating a project or module
- Adding CDI support for an existing module
- Changing the CDI version

Making sure that the CDI Support plugin is enabled

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, select Plugins.
3. In the right-hand part of the dialog, on the Plugins page , type `cd` in the search box. As a result, only the plugins whose names and descriptions contain `cd` are shown in the list of plugins.
4. If the check box to the right of Java EE: Context and Dependency Injection is not selected, select it.
5. Click OK in the Settings dialog.
6. If suggested, restart IntelliJ IDEA.

Enabling CDI support when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java Enterprise . In the right-hand part of the page, specify the JDK to be used and select the Java EE version to be supported.
3. Under Additional Libraries and Frameworks, select the CDI: Context and Dependency Injection check box.
4. You'll need a library that implements CDI. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library.
     To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the library files that implement CDI. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

   Click Next.
5. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Adding CDI support for an existing module

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the module of interest and select Add Framework Support.
3. In the left-hand pane of the Add Frameworks Support dialog that opens, select the CDI: Context and Dependency Injection check box.
4. You'll need a library that implements CDI. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library.
     To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the library files that implement CDI. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

5. Click OK in the Add Frameworks Support dialog.

As a result, the CDI library you have specified is added to the list of module dependencies.

Changing the CDI version

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).

2. Select the CDI library whose version you want to change.

3. Click Change Version. (If you are on the Dependencies tab, select to edit the library first.)

4. In the Downloading Options dialog that opens, select the necessary CDI version, specify other settings as needed and click OK.

Copyright

- Copyright
  - Prerequisite
  - Basics
- Associating a Copyright Profile with a Scope
- Generating and Updating Copyright Notice

Prerequisite

Before using copyright profiles and generating copyright notices, make sure that the Copyright plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Basics

Your project can contain files to be protected by copyright. Instead of generating and editing copyright notices in each file separately, IntelliJ IDEA suggests that you use copyright profiles. Each copyright profile is associated with a certain scope and defines the text and formatting of the copyright notice to be inserted in the files within this scope.

You may need several different copyright notices for files within the same project. In this case, IntelliJ IDEA enables you to define several profiles and associate them with relevant scopes.

You can also declare one of the available profiles as default project copyright profile. The settings of this profile will be applied to any project file that is not included in any scope or belongs to a scope, which has no associated copyright profile .

A copyright profile can contain an explicit plain text of the copyright notice or its definition through a Velocity template. You can type the desired text or a Velocity template manually or import a copyright notice definition from an existing profile.

Note the following:

- If a file does not belong to any scope with a copyright profile assigned, the default copyright profile is used.
- If the No copyright option has been set as the Default Copyright Profile in the Copyright settings dialog box, no copyright notice will be created or updated and IntelliJ IDEA will prompt you to configure a profile to apply.

Associating a Copyright Profile with a Scope

On this page:

– Associating a Profile with a Scope

– Setting the Default Copyright Profile for a Project

Associating a Profile with a Scope

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Copyright under Editor. The Copyright page that opens displays mappings between the available copyright profiles and scopes.
   Note that unless you have at least one copyright profile and scope , the controls of the Scope-profile combinations area are disabled.
2. Click + (`Alt+Insert`). A row to define a new mapping is created.
3. From the Scope drop-down list, select the desired scope.
   To define a new scope, open the Scopes page or click the link in the lower part of the Copyright page.
4. From the Copyright drop-down list, select the profile to apply to the selected scope.
5. Continue composing the list of scope-profile mappings. Use + (`Alt+Insert`) or – (`Alt+Delete`) to add or remove the items. To reorder the mappings, use ↑ (`Alt+Up`) and ↓ (`Alt+Down`).

Setting the Default Copyright Profile for a Project

The settings of the default profile will be applied to any project file that is not included in any scope or belongs to a scope, which has no associated copyright profile.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Copyright under Editor.
2. On the Copyright page that opens, select the desired copyright profile from the Default project copyright drop-down list.
   Note that is you select No copyright, any attempt to generate a copyright notice will fail and IntelliJ IDEA will prompt you to create a profile to apply.

Generating and Updating Copyright Notice

In this section:

- Prerequisite
- Creating a Copyright Profile from Scratch
- Creating a Copyright Profile Based on the Settings of an Existing Profile
- Importing an Existing Copyright Notice Text
- Discarding a Copyright Profile
- Adding a Copyright Notice
- Updating a Copyright Notice

Prerequisite

Before using copyright profiles and generating copyright notices, make sure that the Copyright plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Creating a Copyright Profile from Scratch

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click Copyright Profiles under Copyright. The Copyright Profiles page opens.
2. Click the Add button on the toolbar in the Create new copyright profile dialog box that opens specify the name of the new profile and click OK. You return to the Copyright Profiles page where the new profile is added to the list of available copyright profiles.
3. Specify the copyright notice to be generated. Do one of the following:
   - Type the desired plain text.
   - Define a Velocity template , then click the Validate button to check that the template has been specified correctly. Find the list of supported Velocity variables in the Copyright Profiles dialog box reference.
4. In the Regexp to detect copyright in comments text box, type a character string to distinguish copyright notices from other comments during copyright update.

> **Tip** Make sure that this regular expression matches the above specified copyright notice. Otherwise instead of updating copyright notices, IntelliJ IDEA will insert new ones.

Creating a Copyright Profile Based on the Settings of an Existing Profile

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click Copyright Profiles under Copyright. The Copyright Profiles page opens.
2. Select the desired profile to inherit the settings from and click the Copy button on the toolbar.
3. In the Create new copyright profile dialog box that opens specify the name of the new profile and click OK. You return to the Copyright Profiles page where the new profile is added to the list of available copyright profiles.
4. View and edit the profile settings. Proceed as during creation of a profile from scratch .

Importing an Existing Copyright Notice Text

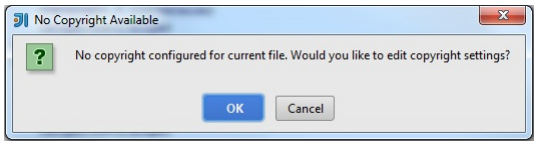
1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click Copyright Profiles under Copyright. The Copyright Profiles page opens.
2. Click the Import button on the toolbar.
3. In the dialog that opens , choose the location of the `%.ipr%` file that refers to the copyright profile with the desired notice definition.
4. From the Choose profile to import list, that appears, select the desired copyright profile.
5. In the Create new copyright profile dialog box that opens specify the name of the new profile and click OK. You return to the Copyright Profiles page where the new profile is added to the list of available copyright profiles.
6. Proceed as during creation of a profile from scratch .

Discarding a Copyright Profile

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click Copyright Profiles under Copyright. The Copyright Profiles page opens.
2. Select the profile you want to discard and click the Delete button on the toolbar.

Adding a Copyright Notice

1. Open the desired file in the editor.
2. Press `Alt+Insert` .
3. From the list that is displayed in the editor, choose Copyright.
4. The following steps depend on the belonging of a file to a certain scope and the value of the Default project copyright field in the Copyright page.
   - If the file where you want to generate a copyright notice belongs to a certain scope, the copyright notice is generated according to the copyright profile settings.
   - If the field Default project copyright has the value No copyright and the file where you want to generate a copyright notice does not belong to any scope, IntelliJ IDEA prompts you to configure copyright settings:



   Click OK to open the Copyright page and configure the desired settings.

Note that if a file belongs to several scopes, these scopes are checked upside down. So doing, the first scope with this file is declared the proper one, and its copyright profile is used.

Updating a Copyright Notice

1. In the Project tool window, select files or directories where you want the copyright notice to be updated or just open the desired file in the editor.
2. Right-click the selection, and choose Update Copyright on the context menu.
   To have copyright notices updated correctly, make sure that the keyword specified in the copyright profile is a part of the notice definition. Otherwise, IntelliJ IDEA will not detect copyright notices and, instead of updating existing notices, will insert new ones.

Tip You can also update copyright notices when committing changes .

Databases and SQL

In this section:

- Databases and SQL
  - Databases and SQL support
- Managing Data Sources
- Working with the Database Tool Window
- Working with Database Consoles
- Working with the Table Editor
- Running SQL Script Files
- Running Injected SQL Statements

Databases and SQL support

IntelliJ IDEA features for working with databases and SQL include:

- Integration with the most popular database management systems such as Oracle , PostgreSQL , MySQL , SQL Server and others. To be able to work with your databases, you should define them as data sources. See Managing Data Sources .
- Database tool window for managing data structures in your databases (View | Tool Windows | Database). See Working with the Database Tool Window .
- Database consoles that let you compose and execute SQL statements as well as analyze and modify retrieved data (`Ctrl+Shift+F10` in the Database tool window). See Working with Database Consoles .
- Table editor that provides a GUI for working with table data (`F4` in the Database tool window). See Working with the Table Editor .
- SQL code generation and editing features in the database consoles and the editor, e.g.
  - Predefined code snippets (a.k.a. live templates) such as for `CREATE TABLE` , `SELECT` , `INSERT` , `UPDATE` and other statements (`Ctrl+J`).
  - Auto-completion and highlighting of SQL keywords, and table and column names.
  - Data type prompts for columns (`Ctrl+P`).

  Standardized and DBMS-specific SQL dialects are supported.
- Structure view for tables in the table editor and Database Console tool window as well as for open database consoles and SQL files (`Ctrl+F12`). See e.g. Using the Structure view to sort data, and hide and show columns.
- Quick documentation view for database objects and table cells (`Ctrl+Q`). See e.g. Using the quick documentation view.
- Navigation capabilities, e.g.
  - From a table or column reference to its definition: `Ctrl+B` .
  - To the view of a table or column in the Database tool window: `Alt+F1` | Database View.
  - By means of the navigation bar: `Alt+Home` .
  - By means of the Switcher: `Ctrl+Tab` .
- Database diagrams (`Ctrl+Alt+U` or `Ctrl+Shift+Alt+U` in the Database tool window).

Managing Data Sources

On this page:

- Data sources
- Defining a database as a data source
- Creating a DB data source for H2 or SQLite by means of drag and drop
- Creating DB data sources by importing connection settings
- Creating a DDL data source
- Creating a DDL data source by means of drag and drop
- Changing data source settings
- Removing data sources

Data sources

To be able to work with your databases in IntelliJ IDEA, you should define them as data sources.

In addition to data sources that correspond to real databases (DB data sources), IntelliJ IDEA also supports DDL data sources. These are represented by one or more SQL files containing data definition language statements (SQL DDL statements).

Metaphorically, DDL data sources function as databases without data.

Data sources provide the basis for SQL coding assistance and code validation.

Defining a database as a data source

1. Open the Database tool window and click on the toolbar.
2. In the Data Sources and Drivers dialog that opens, click + and select the database management system (DBMS) that you are using.
3. In the Name field, if necessary, edit the name of the data source.
4. If there is the message Download missing driver files in the lower part of the dialog, specify the driver files.
   (To interact with a database, IntelliJ IDEA needs a database driver. The drivers, generally, are DBMS-specific.)

   Do one of the following:
   - To download the necessary driver, click the Download link.
   - To specify the driver files that you already have available on your computer, click the <DriverName> link to the right of Driver.
     On the page where the driver settings are shown, under Driver files, click + and select the files in the dialog that opens.

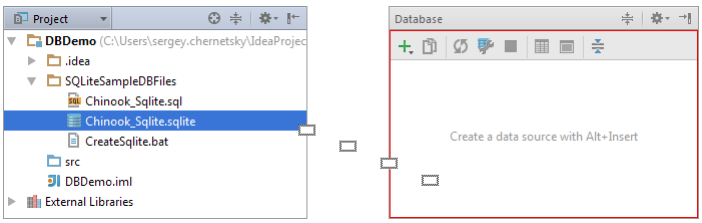
     Go back to the page with the data source settings.
5. Specify the database settings.
   For most of the DB management systems, these are the database host name (or IP address), port, the database name, and also your database user name and password.

   If your database is a local file or directory (which may be the case for SQLite, H2, Derby or HSQLDB), you should specify the location of that file or directory. To do that, click ... to the right of the File or Path field and select the database file or directory in the dialog that opens.
6. To access your database using SSH or SSL, select the SSH/SSL tab and specify the corresponding settings there.
7. To make sure that the settings are correct and IntelliJ IDEA can properly interact with your database, click Test Connection on the General tab.
8. If the settings you have specified identify more than one database or schema, you can select the database or schema you are going to work with. To do that, use the Schemas tab .
9. Click OK to save the settings and close the dialog.
   As a result, your new data source is shown in the Database tool window. The input pane of the associated database console opens to the right of the tool window.

Creating a DB data source for H2 or SQLite by means of drag and drop

If you have H2 or SQLite database files available locally, you can create DB data sources for them by dragging the files to the Database tool window. The files can be dragged from the Project tool window, or from your file manager (e.g. Explorer or Finder).

1. If the database files are in your project folder, open the Project tool window. Otherwise, open your file manager.
2. Open the Database tool window .
3. Select the file or files of interest in the Project tool window, or in your file manager.
4. Drag the selected file or files into the Database tool window. (For each of the files a separate data source will be created.)



5. If you don't have the necessary database driver files yet, you can download them now. Click on the toolbar of the Database tool window. (Alternatively, select Properties from the context menu.)
6. In the Data Sources and Drivers dialog that opens, within the line Driver files missing, click the Download link.
7. Click Test Connection to make sure that IntelliJ IDEA can properly communicate with the database.
8. Click OK in the Data Sources and Drivers dialog.

Creating DB data sources by importing connection settings

XML files that contain database connection settings can be used for creating DB data sources. These may be Spring, Hibernate, JPA and Tomcat `context.xml` configuration files.

1. If the files that you want to import the settings from are not in your project yet, copy them there.
2. Open the Database tool window .
3. Do one of the following:
   - Click + on the toolbar and select Import from sources.
   - Right-click the area under the toolbar or any of the existing data sources, point to New and click Import from sources.

   The Data Sources and Drivers dialog opens. The names of candidate data sources are shown in the left-hand pane in green.
4. Specify the driver files if they are missing.
   Do one of the following:
   - To download the necessary driver, click the Download link.
   - To specify the driver files that you already have available on your computer, click the <DriverName> link to the right of Driver.
     On the page where the driver settings are shown, under Driver files, click + and select the files in the dialog that opens.

     Go back to the page with the data source settings.
5. Click Test Connection to make sure that IntelliJ IDEA can properly communicate with the database.
6. Click OK in the Data Sources and Drivers dialog.

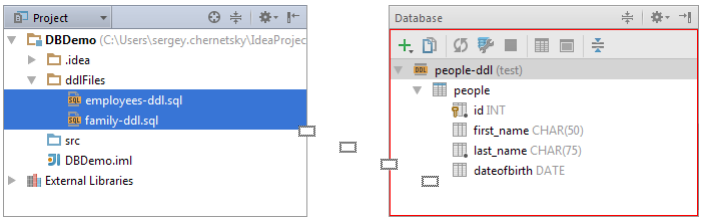
Creating a DDL data source

1. Open the Database tool window and click on the toolbar.
2. In the Data Sources and Drivers dialog that opens, click + and select DDL Data Source.
3. In the Name field, if necessary, edit the name of the data source.
4. Under DDL Files, click + and select the necessary SQL file or files in the dialog that opens.
5. From the Extend list, if necessary, select another data source as a parent. As a result, the data source whose properties you are editing will "inherit" all the DDL definitions from its parent.
6. Click OK to save the settings and close the dialog.

Creating a DDL data source by means of drag and drop

You can create DDL data sources by dragging DDL SQL files to the Database tool window. The files can be dragged from the Project tool window, or from your file manager (e.g. Explorer or Finder).

1. If the necessary DDL SQL files are in your project folder, open the Project tool window. Otherwise, open your file manager.
2. Open the Database tool window .
3. If you already have one or more DDL data sources defined but there are no DB data sources, expand one of the data source nodes in the Database tool window.
4. Select the file or files of interest in the Project tool window, or in your file manager.
5. Drag the selected file or files into the Database tool window. For a new data source to be created, the red border, when dropping the file or files, should surround most of the window area (rather than one of the existing data sources).



Changing data source settings

1. Open the Database tool window and select the data source of interest.
2. Do one of the following:
   - Click on the toolbar.
   - Select Properties from the context menu.
   - Press `Alt+Enter` .
3. In the Data Sources and Drivers dialog that opens, edit the settings as necessary. See:
   - DB data source settings
   - DDL data source settings

Removing data sources

To remove unnecessary data sources, you can use the Database tool window or the Data Sources and Drivers dialog.

Using the Database tool window. Select the data sources to be removed and do one of the following:

- Press `Delete` .
- Select Delete from the context menu.
- Select Edit | Delete.

Using the Data Sources and Drivers dialog. Select the data sources to be removed and do one of the following:

- Click − on the toolbar.
- Press `Delete` .

– Select Remove from the context menu.

Working with the Database Tool Window

The Database tool window provides access to functions for working with databases and DDL data sources. It lets you view and modify data structures in your databases, and perform other associated tasks.

- Opening the Database tool window
- Adjusting the view
- Creating a data source
- Synchronizing the view of a DB data source
- Finding items
- Finding occurrences of an item name
- Creating a copy of a data source
- Creating a table, a column, an index, or a primary or foreign key
- Viewing basic info about an item
- Renaming a table or column
- Modifying the definition of a table, column, index or a foreign key
- Previewing changes
- Removing items
- Opening a default database console
- Creating and opening a new database console
- Opening the table editor
- Generating DDL definitions for data sources, tables and columns
- Comparing table structures
- Saving table data in a file
- Viewing diagrams
- Closing database connections

See also, Database Tool Window .

Opening the Database tool window

Do one of the following:

- Select View | Tool Windows | Database.
- Point to or in lower-left corner of the workspace, and then click Database.
- Click Database on the right-hand tool window bar (if the tool window bars are currently shown).

Adjusting the view

You can adjust the view in the tool window by turning the corresponding view options on and off. To access those options, click on the title bar.

For more information, see View options.

Creating a data source

To start creating a data source , you can use the New command when the Database tool window is active, e.g.:

- File | New
- on the toolbar
- New in the context menu
- `Alt+Insert`

The DDL Data Source option is for creating a DDL data source. Other data source options correspond to different scenarios of creating a DB data source:

- Data Source. A "usual way" of creating a data source. In this scenario, you start by selecting your DBMS.
- Data Source from URL. In this scenario, you start by specifying your database URL.
- Data Source from Path. In this scenario, you start by specifying your database location (a local file or folder). This option is appropriate only for Derby, H2, HSQLDB and SQLite.

Alternatively, you can open the Data Sources and Drivers dialog (e.g.) and use the Add command there: Add from the context menu, on the toolbar, or `Alt+Insert`.

For more information, see Managing Data Sources .

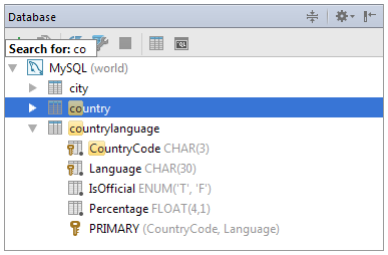
Synchronizing the view of a DB data source

If the Auto sync option for a DB data source if off, the only way to synchronize its view in the Database tool window with the actual state of the database is by using the Synchronize command.

1. Select the item whose view you want to synchronize. This may be a DB data source, schema or table.
2. Do one of the following:
   - Press `Ctrl+Alt+Y`.
   - Click on the toolbar.
   - Select Synchronize from the context menu.

Finding items

To find an item of interest, simply start typing its name. The specified text within item names is highlighted, and the first of the items that contains the specified text is selected.



Finding occurrences of an item name

To find occurrences of the name of an item (e.g. table or column) in your files and consoles:

1. Select the item of interest.
2. Do one of the following:
   - Press `Alt+F7`.
   - Select Find Usages from the context menu.
   - Select Edit | Find | Find Usages in the main menu.

Creating a copy of a data source

1. Select the data source of interest.
2. Do one of the following:
   - Click on the toolbar.
   - Select Duplicate from the context menu.
   - Press `Ctrl+D`.

Creating a table, a column, an index, or a primary or foreign key

1. Depending on what you are going to create:
   - To create a table, select a schema, table or column within the target DB data source.
   - To create a column, select the target table or a column within that table.
   - To create an index, or a primary or foreign key, select the column or columns for which you want to create an index, or a primary or foreign key constraint.
2. Carry out the New command and select the item to be created. E.g. for a table, do one of the following:
   - Select File | New | Table.
   - Click + and select Table.
   - In the context menu, select New | Table.
   - Press `Alt+Insert` and select Table.
3. In the dialog that opens, specify the item definition.

Viewing basic info about an item

You can view basic info about an item in the quick documentation view. For a table, for example, the first ten rows and the table definition (the `CREATE TABLE` statement) are shown.

To open the quick documentation view, select the item of interest and do one of the following:

- Select View | Quick Documentation.
- Press `Ctrl+Q`.

See also, Allow database access.

Renaming a table or column

1. Select the table or column to be renamed.
2. Do one of the following:
   - Select Refactor | Rename.
   - Select Rename from the context menu.
   - Press `Shift+F6`.
3. Use the dialog that opens to specify a new name and associated options.

Modifying the definition of a table, column, index or a foreign key

1. Select the item whose definition you want to change. This may be a table, a column, an index or a foreign key.
2. Do one of the following:
   - Select Modify <item_type> from the context menu (e.g. Modify Table).
   - Press `Ctrl+F6`.
3. Use the dialog that opens to change the item definition.

Previewing changes

Changes to database objects sometimes assume associated changes to SQL script files and statements in database consoles. For example, you may be changing the name of a table, and this name may be used in your files and consoles.

In such cases, you can look at potential changes, and decide where those changes are desirable and where aren't.

Potentially affected code fragments are shown in the Find tool window when you click Preview in the corresponding dialogs. Here is an overview of some of the available

controls:

- Exclude (`Delete`) and Remove (`Alt+Delete`). Use these context menu commands for the items that shouldn't be changed.
- Execute SQL Script. If this option is on, and you click Do Refactor, the corresponding SQL statements are run to modify the corresponding database objects.
- Open in Console. Use this button to open the corresponding SQL statements in a database console .
- Do Refactor. Click this button to change the corresponding code fragments and, if the Execute SQL Script option is on, to run the corresponding SQL statements.

Removing items

To remove data sources, use the Remove command (Edit | Remove, Remove from the context menu, or `Delete` on the keyboard).

To remove all other item types such as tables, columns and indexes, use the Drop command (Edit | Drop, Drop from the context menu, or `Delete` on the keyboard).

For removing primary key constraints, in addition, there is the Drop Associated | Primary Key context menu command.

See also, Confirm Drop dialog .

Opening a default database console

1. Select the DB data source of interest or any node within it.
2. Do one of the following:
   - Click on the toolbar.
   - Select Open Console from the context menu.
   - Press `Ctrl+Shift+F10` .

For more information, see Working with Database Consoles .

Creating and opening a new database console

1. Select the DB data source of interest or any node within it.
2. Do one of the following:
   - Select Open New Console from the context menu.
   - Click and select Console File.

For more information, see Working with Database Consoles .

Opening the table editor

1. Select the table of interest.
2. Do one of the following:
   - Click on the toolbar.
   - Select Table Editor from the context menu.
   - Press `F4` .

For more information, see Working with the Table Editor .

Generating DDL definitions for data sources, tables and columns

1. Select the data source, table or column of interest.
2. Do one of the following:
   - Select Copy DDL from the context menu.
   - Press `Ctrl+Shift+C` .

   The definitions are generated on the clipboard and you can paste them into a database console or an SQL file.

Comparing table structures

1. Select two data sources, schemas or tables.
2. Do one of the following:
   - Select Compare from the context menu.
   - Press `Ctrl+D` .

The comparison results are shown in the differences viewer .

Saving table data in a file

1. Right-click the table or view of interest.
2. Point to Dump Data To File and select the output format (e.g. Comma Separated Values (CSV)).
3. In the dialog that opens, specify the file location and name.

Viewing diagrams

To open a diagram for a data source, schema or table, select the item of interest and do one of the following:

- Press `Ctrl+Shift+Alt+U` or `Ctrl+Alt+U` .
- In the context menu, select Diagrams, and then select Show Visualisation or Show Visualisation Popup.

Closing database connections

IntelliJ IDEA connects to databases automatically, when needed. (The names of the data sources with open database connections are shown in the Database tool window in bold.)

To close unnecessary database connections, select the corresponding data sources and do one of the following:

- Click on the toolbar.
- Select Disconnect from the context menu.
- Press `Ctrl+F2`.

Working with Database Consoles

Database consoles let you compose and execute SQL statements for databases defined in IntelliJ IDEA as data sources. They also let you analyze and modify the retrieved data.

The following standardized and DBMS vendor-specific SQL dialects are supported: DB2, Derby, H2, HSQLDB, MySQL, Oracle, Oracle SQL*Plus, PostgreSQL, SQL Server, SQL92, SQLite, and Sybase .

One database console is created for a data source automatically when a data source is created. If necessary, you can create additional consoles.

Database consoles are persistent: they are stored as SQL files.

A database console created in one of your projects can then be accessed from any other project.

General console usage instructions are on this page:

- Creating a database console
- Opening a database console
- Viewing and modifying console settings
- Changing the SQL dialect
- Closing a console
- Managing database consoles

More specific subjects are discussed in:

- Writing and Executing SQL Statements
- Working with Query Results

See also, Database Console .

Creating a database console

When you create a DB data source, one database console for that data source is created automatically. If necessary, you can create additional consoles.

To create a database console, you can use the Database tool window or the Scratches view of the Project tool window . The procedure is the same in both cases:

Select the data source of interest or any node within it, and do one of the following:

- Select File | New | Console File from the main menu.
- Select New | Console File from the context menu
- Press `Alt+Insert` and select Console File.

In the Database tool window, you can also use + | Console File and the Open New Console context menu command.

Opening a database console

The Database tool window lets you open only default consoles, i.e. the ones that were created by IntelliJ IDEA automatically.

To open the consoles that you created yourself, you should use the Scratches view of the Project tool window .

Using the Database tool window. Select the data source of interest or any node within and do one of the following:

- Click on the title bar if the toolbar is hidden.
- Click on the toolbar if the toolbar is shown.
- Select Open Console from the context menu.
- Press `Ctrl+Shift+F10`.

Using the Scratches view. For the console of interest, do one of the following:

- Double-click the console.
- Select View | Jump to Source from the main menu.
- Select Jump to Source from the context menu.
- Press `F4`.

Viewing and modifying console settings

Before actually starting to use a console, you may want to take a look at the console settings and adjust them to your needs.

- To access these settings, click on the toolbar of the input pane or on the toolbar of the Database Console tool window. (Alternatively, `Ctrl+Alt+S` | Tools | Database.)

As a result, the Database page of the Settings / Preferences dialog will open.

Changing the SQL dialect

By default, the SQL dialect used in a console is defined by the DBMS of an associated data source. If for some reason you want to use a different dialect:

- Right-click the editing area of the input pane, select Change Dialect (<CurrentDialect>), and select the necessary dialect.
  In addition to particular dialects, also the following option is available:
  - <Generic>. Basic SQL92-based support is provided including completion and highlighting for SQL keywords, and table and column names. Syntax error highlighting is not available. So all the statements in the input pane are always shown as syntactically correct.

Closing a console

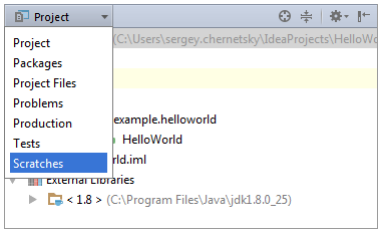
1. Click (`Ctrl+Shift+F4`) to close the Database Console tool window.

2. Click ✕ on the editor tab (`Ctrl+F4`) to close the input pane.

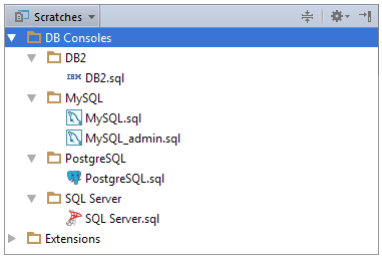
Managing database consoles

To manage your database consoles, use the Scratches view of the Project tool window .

To open this view, select Scratches from the list in the left-hand part of the title bar.



The view shows the existing database consoles (represented by SQL files) grouped by your data sources (shown as folders). The default consoles (the ones that were created by IntelliJ IDEA automatically) have the same names as the corresponding data sources.



You can:

- Create new consoles. Select the target data source or a node within it, and do one of the following:
  - Select File | New | Console File from the main menu.
  - Select New | Console File from the context menu
  - Press `Alt+Insert` and select Console File.
- Rename your consoles. When a new console is created, it has the name of the data source with a number at the end, e.g. `MySQL_1` . If you want to give a console a more descriptive name, select the console and do one of the following:
  - Select Refactor | Rename from the main or the context menu.
  - Press `Shift+F6` .

  Then, specify a new name in the dialog that opens.
- Save your console files in arbitrary directories. Select the console and then select Refactor | Copy (`F5`). Specify the file name and location in the dialog that opens.
- Group your consoles. This is done by creating folders and then dragging your consoles into those folders.
- Open your consoles. Select the consoles of interest and do one of the following:
  - Select View | Jump to Source from the main menu.
  - Select Jump to Source from the context menu.
  - Press `F4` .
- View the history of changes for your consoles. Select File | Local History | Show History from the main menu or Local History | Show History from the context menu.
- Delete individual consoles and groups of consoles. Use Edit | Delete, Delete from the context menu or `Delete` on the keyboard.

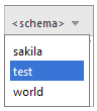
Writing and Executing SQL Statements

Use the input pane of the database console to compose and execute your SQL statements, and also to perform other, associated tasks. (The input pane is shown as a tab in the editor.)

- Selecting the default schema or database
- Composing SQL statements
- Editing data for INSERT statements in table format
- Navigating to a table or column view in the Database tool window
- Configuring the Execute command
- Executing an SQL statement
- Executing parameterized statements
- Executing a group of statements
- Executing all statements
- Executing a part of a statement (e.g. a subquery)
- Executing auto-memorized statements
- Outputting the result of a SELECT statement into a file
- Using the error notification bar
- Canceling running statements
- Managing database transactions
- Showing execution plans
- Showing DBMS_OUTPUT for Oracle

Selecting the default schema or database

You can select the default schema or database by using the list in the right-hand part of the toolbar. If you do so, you'll be able to omit the name of that schema or database in your statements.

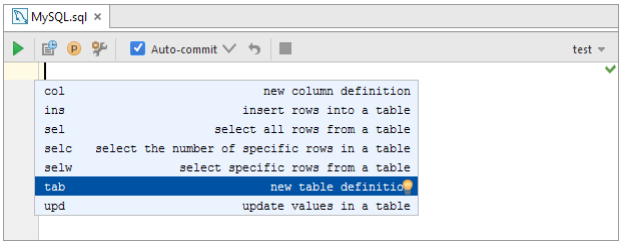


(You cannot switch schemas like this for read-only MySQL data sources, see Read-only .)

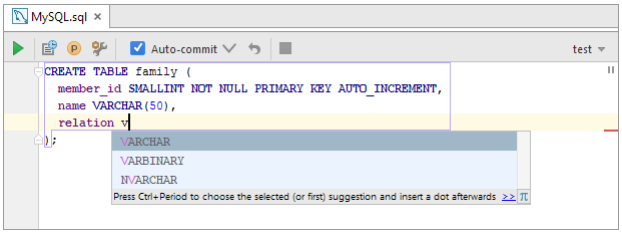
Composing SQL statements

When composing your SQL statements, use:

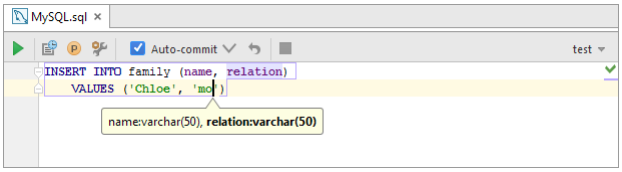
- Predefined patterns (Ctrl+J or Code | Insert Live Template).



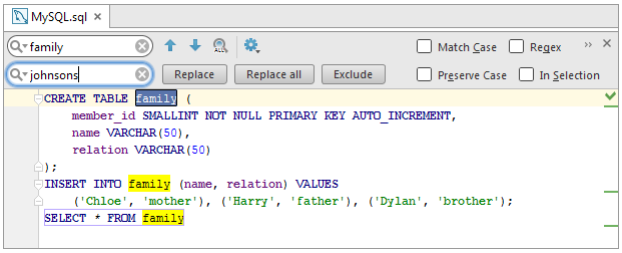
- Auto-completion and highlighting of SQL keywords, and table and column names.



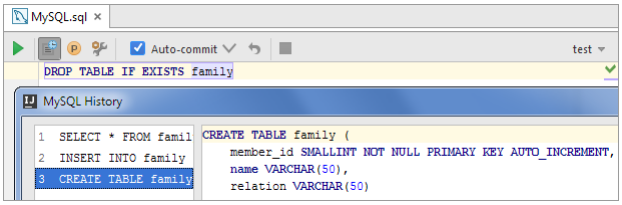
- Data type prompts for columns (Ctrl+P or View | Parameter Info).



- Advanced find and replace capabilities (Ctrl+F or Edit | Find | Find, and Ctrl+R or Edit | Find | Replace).



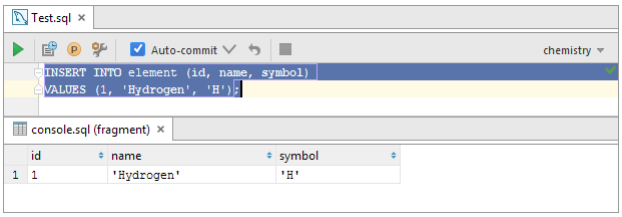
- The console history (or Ctrl+Alt+E). See Executing auto-memorized statements.



See also, Navigating to a table or column view in the Database tool window.

Editing data for INSERT statements in table format

1. Select the `INSERT` statement of interest.
2. Select Edit as Table from the context menu. As a result, the table editor opens.



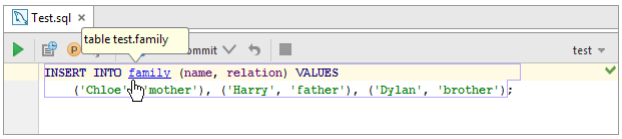
3. Use context menu commands and associated shortcuts for working with the data in the table editor.

Navigating to a table or column view in the Database tool window

When composing a statement, it's sometimes useful to take a look at the structure of a table, or to see the info about a column (field) in the context of the table to which it belongs. For such purposes, IntelliJ IDEA provides the ability to switch from the name of a table or column in the input pane to its view in the Database tool window.

The following ways are available for using this feature:

- Place the cursor within the name of the table or column of interest. Then use `Ctrl+B`. (Alternatively, you can use Navigate | Declaration from the main menu or Go To | Declaration from the context menu.)
- Press and hold the `Ctrl` key, and point to the name of interest. When the text turns into a hyperlink, click the hyperlink.



Configuring the Execute command

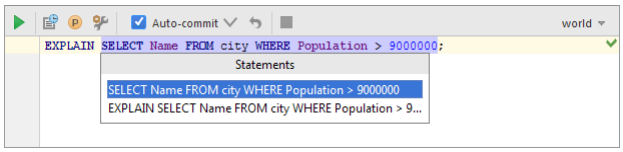
The Execute command (on the toolbar, `Ctrl+Enter` or Execute from the context menu) is used to run your statements.

IntelliJ IDEA provides many options for the Execute command depending on the cursor position and on whether there is a selection.

The options are specified on the Tools | Database page in the Settings / Preferences dialog (File | Settings | Tools | Database on Windows and Linux; IntelliJ IDEA | Preferences | Tools | Database on OS X). For more information, see Execute in Console.

Executing an SQL statement

1. Place the cursor within the statement.
2. Do one of the following:
   - Click on the toolbar.
   - Press `Ctrl+Enter`.
   - Select Execute from the context menu.
3. Select the statement or statements to be run. (The suggestion list always contains an item for running all the statements.)

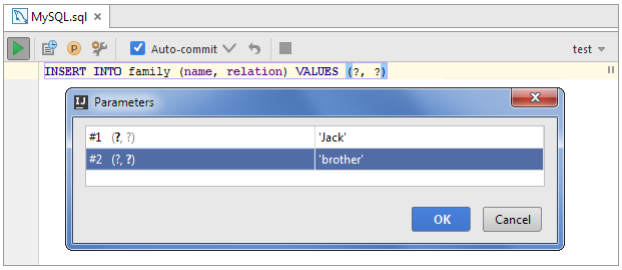


See also, Execute in Console.

Executing parameterized statements

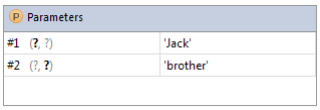
Your statements can contain parameters, however, by the time you execute such statements the values of the parameters must be specified. There are the following ways of specifying the parameter values:

- Click on the toolbar or press `Ctrl+Enter` to execute the statement. In the dialog that opens, specify the parameter values and click OK.



(To start editing a value, switch to the corresponding table cell and start typing. To indicate that you have finished editing a value, press `Enter` or switch to a different cell. To quit the editing mode and restore an initial value, press `Escape`.)

– Alternatively, you can open the Parameters pane in the Database Console tool window (on the toolbar) and specify the corresponding values there. (The values are edited in the same way as in the corresponding dialog.) Then execute the statement (on the toolbar or `Ctrl+Enter`).



For more information, see Parameters pane.

See also, User Parameters and Always review parameters before execution.

Executing a group of statements

To execute a group of statements that follow one another in the console, select (highlight) the statements (to select all the statements, use `Ctrl+A`) and do one of the following:

– Click on the toolbar.
– Press `Ctrl+Enter`.
– Select Execute from the context menu.

See also, Using the error notification bar and Execute in Console.

Executing all statements

To execute all the statements contained in a console, as an alternative to the Execute command, you can use the Run console.sql command.

This command is available in the context menu, and its keyboard equivalent is `Ctrl+Shift+F10`.

The Run console.sql command, generally, runs faster but:

– The statements with parameters don't run.
– Retrieved data for the `SELECT` statements are not shown.

Executing a part of a statement (e.g. a subquery)

To execute a part of a statement (e. g. a subquery), select (highlight) the fragment that you want to execute and do one of the following:

– Click on the toolbar.
– Press `Ctrl+Enter`.
– Select Execute from the context menu.

See also, Execute in Console.

Executing auto-memorized statements

As you run SQL statements in the consoles, IntelliJ IDEA memorizes them. So, at a later time, you can view the statements you have already run and, if necessary, run them again.

To open the dialog where the auto-memorized statements are shown (the History dialog), do one of the following:

– Click on the toolbar.
– Press `Ctrl+Alt+E`.

There are two panes in the History dialog. The left-hand pane shows the list of the statements that you have run. For "long" statements, only their beginnings are shown. When you select a statement in this pane, the overall statement is shown in the pane to the right.

You can filter the information: just start typing. As a result, only the statements that contain the typed text will be shown.

You can copy the statements from the History dialog into the input pane of the console. To copy a statement, do one of the following:

– Double-click the statement to be copied.
– Select the statement of interest and press `Enter`.
– Select the statement and click OK.

(Once the statement is in the input pane, you can run it straight away.)

You can delete unnecessary memorized statements. To delete a statement, select the statement in the History dialog and press `Delete`.

Outputting the result of a SELECT statement into a file

Instead of the Result pane of the Database Console tool window, you can output the result of a `SELECT` statement into a file.

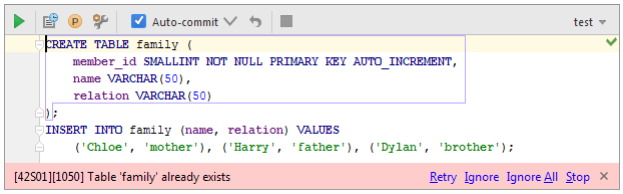
1. Right-click the `SELECT` statement of interest.
2. Point to Execute to File and select the output format.

3. Specify the output file location and name.

Using the error notification bar

If when running a statement an error occurs, an error notification bar appears in the lower part of the input pane.

This bar may be particularly useful when executing a sequence of statements (see Executing a group of statements) because in such a case it lets you select how to react.



The options are:

- Retry. Execute the sequence of statements starting from the one that caused the error.
- Ignore. Skip the erroneous statement and execute the sequence starting from the next statement. If another error occurs, the error notification bar will appear again.
- Ignore All. Skip the erroneous statement and execute the sequence starting from the next statement. If other errors occur, all the erroneous statements will be skipped and the error notification bar won't appear for these statements.
- Stop. Stop the execution of the sequence.

Showing the error notification bar in the input pane is enabled or disabled in the Settings dialog (the Show error notifications in editor check box on the Database page).

Canceling running statements

To terminate execution of the current statement or statements, do one of the following:

- Click on the toolbar of the input pane, or on the toolbar of the Dababase Console tool window.
- Press Ctrl+F2 .

Managing database transactions

The Auto-commit check box, and the Commit and Rollback icons on the toolbar let you manage database transactions.

The Auto-commit check box is used to turn the autocommit mode for the database connection on or off.

In the autocommit mode, each SQL statement is executed in its own transaction that is implicitly committed. Consequently, the SQL statements executed in this mode cannot be rolled back.

If the autocommit mode is off, transactions are committed or rolled back explicitly by means of the Commit ( ) or Rollback ( ) command. Each commit or rollback starts a new transaction which provides grouping for a series of subsequent SQL statements.

In this case, the data manipulations in the transaction scope are committed or rolled back all at once when the transaction is committed or rolled back.

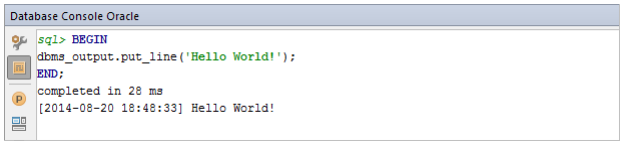
Showing execution plans

The following context menu commands let you show an execution plan (a.k.a. explain plan) for a statement:

- Explain Plan. The result is shown in a mixed tree/table format on a dedicated Plan tab.
- Explain Plan (Raw). The result is shown in table format. (Technically, `EXPLAIN <CURRENT_STATEMENT>` or similar statement is executed.)

Showing DBMS_OUTPUT for Oracle

For Oracle, you can enable or disable showing the contents of the DBMS_OUTPUT buffer in the output pane. To do that, use on the toolbar of the Database Console tool window (Ctrl+F8).



Working with Query Results

When you run a query (a `SELECT` statement) in the console, the data retrieved from the database are shown in table format in the Result pane of the Database Console tool window. Depending on the settings, a new Result tab opens for each query, or one and the same tab is used. In the latter case, the results on the tab are updated for each next query.

Use the Result pane to sort, add, edit and remove the data as well as to perform other, associated tasks.

– Hiding or showing the toolbar
– Pinning the Result tab
– Switching between subsets of rows
– Making all rows visible simultaneously
– Navigating to a specified row
– Navigating to related records
– Sorting data
– Reordering columns
– Hiding and showing columns
– Restoring the initial table view
– Using the Structure view to sort data, and hide and show columns
– Using the quick documentation view
– Transposing the table
– Enabling coding assistance for a column
– Modifying cell contents
– Adding a row
– Deleting rows
– Managing database transactions
– Comparing tables
– Copying table data to the clipboard or saving them in a file
– Copying a selection to the clipboard
– Changing data output format
– Saving a LOB in a file
– Updating the table view
– Viewing the query

Hiding or showing the toolbar

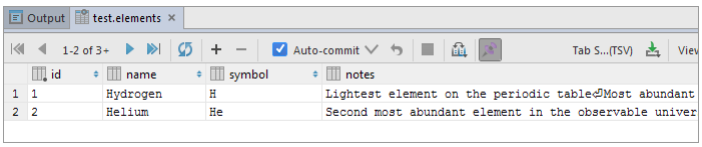
To hide or show the toolbar of the Result pane and also that of the Database Console tool window:

– Click on the title bar of the Database Console tool window and click Show Toolbar.

Pinning the Result tab

If one and the same tab is used to show your query results, and you get the result that you want to keep, you can pin the tab to the tool window. Do one of the following:

– Right-click the tab and select Pin Tab.
– Click on the toolbar.



See also, Show query results in new tab .

Switching between subsets of rows

If only a subset of the rows that satisfy the query is currently shown, to switch between the subsets, use:

– First Page
– Previous Page (`Ctrl+Alt+Up`)
– Next Page (`Ctrl+Alt+Down`)
– Last Page

See also, Making all rows visible simultaneously.

Making all rows visible simultaneously

If you want all the rows that satisfy the query to be shown simultaneously:

1. Click on the toolbar of the Database Console tool window.
2. Switch to the Database | Data Views page, specify `0` in the Result set page size field, and click OK.
3. Click or press `Ctrl+F5` to refresh the table view.

See also, Updating the table view and Result set page size .

Navigating to a specified row

To switch to a row with a specified number:

1. Do one of the following:

- Press `Ctrl+G`.
- Right-click the table and select Go To | Row from the context menu.
- Select Navigate | Row from the main menu.

2. In the dialog that opens, specify the row number and click OK.

Navigating to related records

If a row references a record in a different table or is referenced in a different table, you can switch to the corresponding table to see the related record or records.

To switch to a referenced row:

1. Do one of the following:
   - Press `Ctrl+B`.
   - Select Go To | Referenced Data from the context menu.
2. If more than one record is referenced, select the target record in the pop-up that appears.

To switch to a row that references the current one, or to see all the rows that reference the current one:

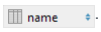
1. Do one of the following:
   - Press `Alt+F7`.
   - Select Go To | Referencing Data from the context menu.
2. Select the target in one of the following categories:
   - First Referencing Row. All the rows in the corresponding table will be shown and the first of the rows that references the current row will be selected.
   - All Referencing Rows. Only the rows that reference the current row will be shown.

The options described above can also be accessed by using one of the following:

- `F4`.
- Go To | Related Data in the context menu.
- Navigate | Related Data in the main menu.

Sorting data

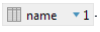
You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: 

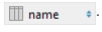
. The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance:

. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: 

.

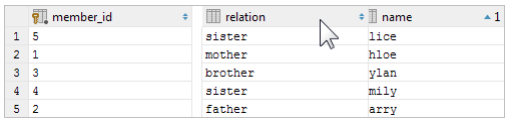
Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: 

.

See also, Restoring the initial table view and Using the Structure view to sort data, and hide and show columns.

Reordering columns

To reorder columns, use drag-and-drop for the corresponding cells in the header row.



	member_id	relation	name 1
1	5	sister	lice
2	1	mother	hloe
3	3	brother	ylan
4	4	sister	mily
5	2	father	arry

See also, Restoring the initial table view.

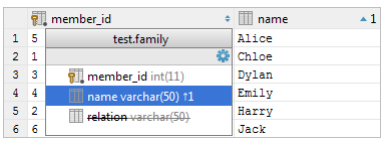
Hiding and showing columns

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
   - Right-click any of the cells in the header row and select Column List.
   - Press `Ctrl+F12`.

   In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

See also, Restoring the initial table view and Using the Structure view to sort data, and hide and show columns.

Restoring the initial table view

Click on the toolbar and select Reset View to restore the initial table view after reordering or hiding the columns, or sorting the data. As a result, the data, generally, becomes unsorted, the columns appear in the order they are defined in the corresponding query, and all the columns are shown.

Using the Structure view to sort data, and hide and show columns

When working with the Result pane, the table structure view is available as the corresponding popup.

The structure view shows the list of all the columns and lets you sort the data as well as hide and show the columns.

To open the structure popup, do one of the following:

– Right-click a cell in the table header row and select Column List.

– Press `Ctrl+F12`.

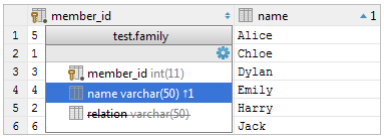
In the popup, select the column of interest and do one of the following:

– To sort the data by this column in the ascending order, press `Shift+Alt+Up`.

– To sort the data in the descending order, press `Shift+Alt+Down`.

– To cancel sorting by this column, press `Ctrl+Shift+Alt+Backspace`.

– To hide the column (or show a hidden column), press `Space`. (The names of hidden columns are shown struck through.)

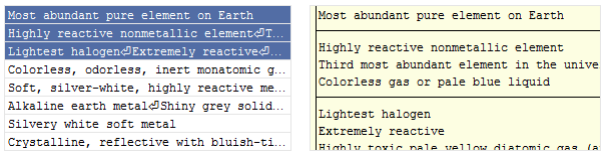


The shortcuts for sorting table data (`Shift+Alt+Up`, `Shift+Alt+Down` and `Ctrl+Shift+Alt+Backspace`) can be used in the Result pane without opening the structure view.

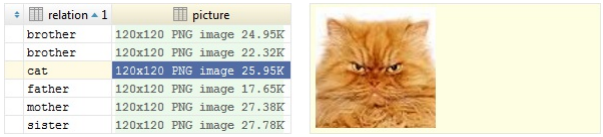
See also, Sorting data, Hiding and showing columns and Restoring the initial table view.

Using the quick documentation view

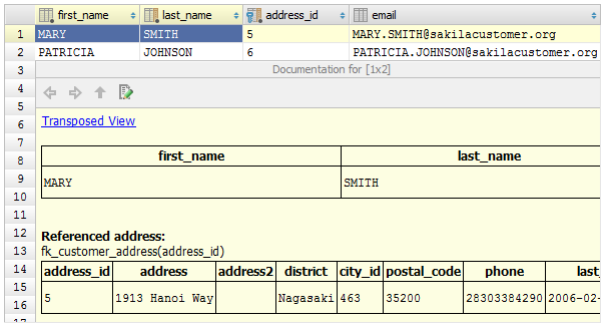
The quick documentation view provides details about the values in the selected cell or cells. For example, if a cell contains long text, normally, you can see only its beginning. The whole text is shown in the quick documentation view.



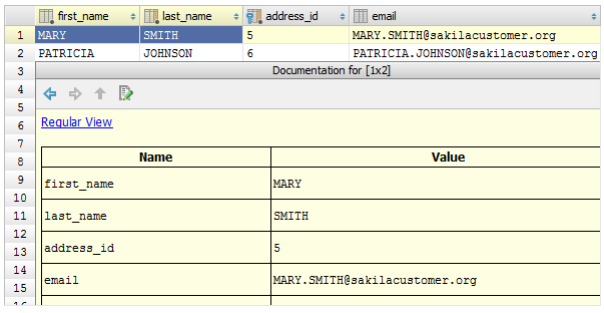
If a cell contains an image, you can see that image in the quick documentation view.



You can also see the records referenced in the current record as well as the records that reference the current one.



If necessary, you can switch to the transposed view. This is when the rows and columns are interchanged. Thus, for a row, the cells are shown one beneath the other.



To open the quick documentation view, press `Ctrl+Q` or select Quick Documentation from the View or the context menu.

To switch to the transposed view, click Transposed View. See also, Transposing the table.

To close the quick documentation view, press `Escape`.

Transposing the table

The transposed table view is available. In this view, the rows and columns are interchanged.

To turn this view on or off, click on the toolbar and select Transpose. Alternatively, use the Transpose context menu command.

Enabling coding assistance for a column

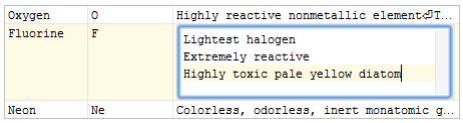
You can assign a column one of the supported languages (e.g. SQL, HTML or XML): right-click the corresponding header cell, select Edit As and select the language. As a result, you get coding assistance for the selected language in all the cells of the corresponding column.

You can also assign a language to an individual cell .

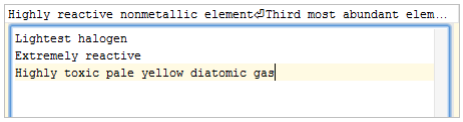
Modifying cell contents

You can modify values in the table cells and, if appropriate, upload files.

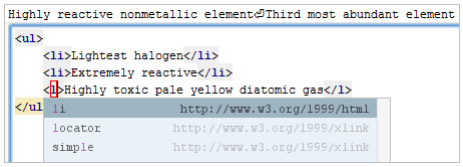
1. To start editing a value or uploading a file, do one of the following:
   - Double-click the corresponding table cell.
   - Right-click the cell and select Edit Value or Edit Maximized from the context menu.
   - Select the cell and press `F2` or `Shift+Enter`. In the latter case, the cell will be maximized.
   - Select the cell and start typing. Note that in this case the initial cell contents are deleted right away and is replaced with the typed value.
2. When in the editing mode, you can:
   - Modify the value right in the cell. To start a new line, use `Ctrl+Enter`. To save the changes, press `Enter`. To restore an initial value and quit the editing mode, press `Escape`.



   - Maximize the cell if you need more room for editing. To do that, press `Ctrl+Shift+Alt+M`, or right-click the cell and select Maximize.
     When working in a maximized cell, use `Enter` to start a new line and `Ctrl+Enter` to save the value. To restore an initial value and quit the editing mode, press `Escape`.



   - Upload a file into the field (e.g. to replace an existing file with a new one). To do that, right-click the cell and select Load File. Then select the necessary file in the dialog that opens.
   - Replace the current value with `null`. To do that, right-click the cell and select Set NULL.
   - Edit a value in the cell as a fragment in one of the supported languages (e.g. SQL, HTML or XML). To do that, right-click the cell, select Edit As and select the language. As a result, you get coding assistance for the language you have selected.



Adding a row

If on the toolbar is enabled, you can add rows to the table.

1. To start adding a row, do one of the following:
   - Click on the toolbar.
   - Right-click the table and select Add New Row from the context menu.
   - Press `Alt+Insert`.

   Note that the context menu Clone Row command (`Ctrl+D`) can be used as an alternative.
2. Enter the values into the cells. For instructions, see Modifying cell contents.
3. To save the new row, select Submit New Row from the context menu or press `Ctrl+Enter`.

Deleting rows

If on the toolbar is enabled, you can delete rows. To do that:

1. Select the row or rows that you want to delete.
   Rows are selected by clicking the cells in the column where the row numbers are shown. To select more than one row, use mouse clicks in combination with the `Ctrl` key.
2. Do one of the following:
   - Click on the toolbar.
   - Press `Ctrl+Y` or `Delete`.
3. Confirm you intention to delete the selected row or rows.

Managing database transactions

The Auto-commit check box, and the Commit and Rollback icons on the toolbar let you manage database transactions.

The Auto-commit check box is used to turn the autocommit mode for the database connection on or off.

In the autocommit mode, each SQL statement is executed in its own transaction that is implicitly committed. Consequently, the SQL statements executed in this mode cannot be rolled back.

(Each change of a value in the table, or adding or deleting a row results in executing an SQL statement.)

If the autocommit mode is off, transactions are committed or rolled back explicitly by means of the Commit ( ) or Rollback ( ) command. Each commit or rollback starts a new transaction which provides grouping for a series of subsequent SQL statements.

In this case, the data manipulations in the transaction scope are committed or rolled back all at once when the transaction is committed or rolled back.

Comparing tables

You can compare the current table with any other table which is open in the table editor or shown in the Database Console tool window. To do that, click on the toolbar and select the table of interest.

The comparison results are shown in the differences viewer .

Copying table data to the clipboard or saving them in a file

1. Do one of the following:
   - Click on the toolbar, and select To File or To Clipboard.
   - Right-click the table, point to Dump Data and select To File or To Clipboard.
2. If you are saving the data in a file, select the output format (the extractor), and then specify the file name and location.

If only a subset of rows is currently shown, all the rows that satisfy the corresponding query are copied to the clipboard or saved in a file anyway.

See also, Changing data output format.

Copying a selection to the clipboard

To copy selected cells or rows to the clipboard, press `Ctrl+C` or select Copy from the main or the context menu.

If the table is shown transposed (see Transposing the table), the range of cells is copied transposed (as shown) if the Allow Transposition option is on and non-transposed (as in the original table) otherwise.

To turn the Allow Transposition option on or off, right-click the table, point to Data Extractor: <current_format> and click Allow Transposition. Alternatively, click Tab S...(TSV) on the toolbar and click Allow Transposition.

Changing data output format

When you copy table data to the clipboard (the Copy and Dump Data | To Clipboard commands), unless you change the format, the data are output as Tab-Separated Values (TSV).

To change the output format or to define a new format for the output:

1. Do one of the following:
   - Click Tab S...(TSV) on the toolbar.
   - Right-click the table and point to Data Extractor: <current_format>.
2. Do one of the following:
   - To select a different output format, click the name of the format.
   - To define a new delimiter-separated values format, or to view or modify the settings for an existing one, click Configure CSV Formats. As a result, the CSV Formats Dialog will open.

Saving a LOB in a file

If a cell contains a binary large object (a.k.a. BLOB or LOB), you can save such a LOB in a file.

1. Right-click the cell that contains the LOB of interest and select Save LOB To File.
2. In the dialog that opens, specify the name and location of the destination file and click OK.

Updating the table view

To refresh the table view, do one of the following:

- Click on the toolbar.
- Right-click the table and select Reload Page from the context menu.
- Press `Ctrl+F5`.

Use this function to:

- Synchronize the data shown with the actual contents of the database.
- Apply the Result set page size setting after its change.

Viewing the query

To see the query that was used to generate the table:

- Click View Query on the toolbar.
  If necessary, you can select the query text and copy it to the clipboard (`Ctrl+C`).

To close the pane where the query is shown, press `Escape`.

Working with the Table Editor

The Table Editor provides a graphical interface for working with table data. You can sort, filter, add, edit and remove the data as well as perform other, associated tasks.

- Opening a table in the Table Editor
- Protecting a table from accidental modifications
- Switching between subsets of rows
- Making all rows visible simultaneously
- Navigating to a specified row
- Navigating to related records
- Sorting data
- Filtering data
- Using quick filtering options
- Reordering columns
- Hiding and showing columns
- Restoring the initial table view
- Using the Structure view to sort data, and hide and show columns
- Using the quick documentation view
- Transposing the table
- Enabling coding assistance for a column
- Modifying cell contents
- Adding a row
- Deleting rows
- Managing database transactions
- Comparing tables
- Copying table data to the clipboard or saving them in a file
- Copying a selection to the clipboard
- Changing data output format
- Saving a LOB in a file
- Updating the table view
- Viewing the query
- Working with the CREATE TABLE statement

Opening a table in the Table Editor

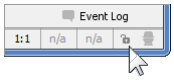
In the Database tool window , do one of the following:

- Double-click the table of interest.
- Click the table and click on the toolbar (if the toolbar is not currently hidden).
- Select the table and press `F4`.
- Right-click the table and select Table Editor from the context menu.

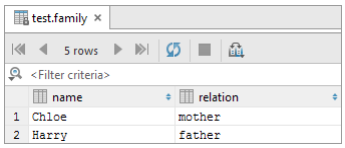
As a result, the table opens in the Table Editor on a separate editor tab.

Protecting a table from accidental modifications

To protect a table from accidental modifications in the Table Editor, you can make it read-only. To do that, click the padlock icon in the lower-right corner of IntelliJ IDEA workspace.



As a result, the icon appearance will change to , a padlock will appear on the corresponding editor tab, and you won't be able to make changes to the table.



To turn off the table's read-only status, click the padlock icon again.

Note that the tables with the read-only status in the Table Editor can still be modified when using the database console or in the Database tool window.

Switching between subsets of rows

If only a subset of all the rows is currently shown, to switch between the subsets, use:

- First Page
- Previous Page (`Ctrl+Alt+Up`)
- Next Page (`Ctrl+Alt+Down`)
- Last Page

See also, Making all rows visible simultaneously.

Making all rows visible simultaneously

If you want all the rows to be shown simultaneously:

- Click ... on the toolbar and select Settings

1. Click on the toolbar and select Settings.
2. Switch to the Database | Data Views page, specify `0` in the Result set page size field, and click OK.
3. Click or press `Ctrl+F5` to refresh the table view.

See also, Updating the table view and Result set page size .

Navigating to a specified row

To switch to a row with a specified number:

1. Do one of the following:
   - Press `Ctrl+G`.
   - Right-click the table and select Go To | Row from the context menu.
   - Select Navigate | Row from the main menu.
2. In the dialog that opens, specify the row number and click OK.

Navigating to related records

If a row references a record in a different table or is referenced in a different table, you can switch to the corresponding table to see the related record or records.

To switch to a referenced row:

1. Do one of the following:
   - Press `Ctrl+B`.
   - Select Go To | Referenced Data from the context menu.
2. If more than one record is referenced, select the target record in the pop-up that appears.

To switch to a row that references the current one, or to see all the rows that reference the current one:

1. Do one of the following:
   - Press `Alt+F7`.
   - Select Go To | Referencing Data from the context menu.
2. Select the target in one of the following categories:
   - First Referencing Row. All the rows in the corresponding table will be shown and the first of the rows that references the current row will be selected.
   - All Referencing Rows. Only the rows that reference the current row will be shown.

The options described above can also be accessed by using one of the following:

- `F4`.
- Go To | Related Data in the context menu.
- Navigate | Related Data in the main menu.

Sorting data

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: name . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance: name 1. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: name 1.

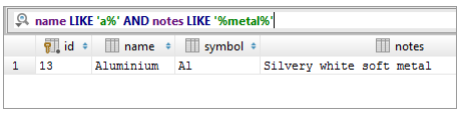
Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: name .

You can turn on the Sort via ORDER BY option , to enable sorting the data by the corresponding DBMS.

See also, Restoring the initial table view and Using the Structure view to sort data, and hide and show columns.

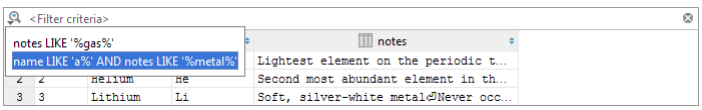
Filtering data

1. If the filter box is not currently shown, click on the toolbar and select Row Filter.
2. In the filter box, specify filtering conditions.
   The filtering conditions are specified as in a `WHERE` clause but without the word `WHERE`, e. g. `name LIKE 'a%' AND notes LIKE '%metal%'`. Within the `LIKE` expressions, the SQL wildcards can be used: the percent sign (`%`) for zero or more characters and underscore (`_`) for a single character.



   To apply the conditions currently specified in the box, press `Enter`. To cancel filtering, delete the contents of the filter box and press `Enter`.

   To reapply a memorized filter, click and select the filter in the list. See also, Filter history size .



   To cancel filtering and hide the filter box, click .

Using quick filtering options

In addition to specifying filtering conditions manually (see Filtering data), you can use quick filtering options.

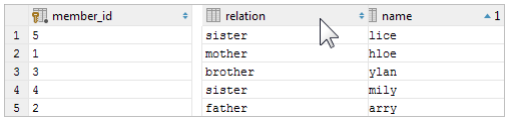
Available as context menu commands, these options are a set of filtering conditions for the current column name. The conditions themselves depend on the value in the current cell.

To use a quick filtering option:

1. Right-click a cell of interest and point to Filter by.
2. Select the necessary condition from the list.

Reordering columns

To reorder columns, use drag-and-drop for the corresponding cells in the header row.



	member_id	relation	name
1	5	sister	lice
2	1	mother	hloe
3	3	brother	ylan
4	4	sister	mily
5	2	father	arry

See also, Restoring the initial table view.

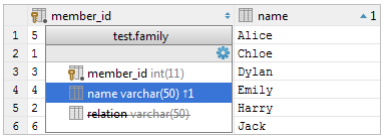
Hiding and showing columns

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
   - Right-click any of the cells in the header row and select Column List.
   - Press `Ctrl+F12`.

   In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

See also, Restoring the initial table view and Using the Structure view to sort data, and hide and show columns.

Restoring the initial table view

Click on the toolbar and select Reset View to restore the initial table view after reordering or hiding the columns, or sorting the data. As a result, the data, generally, becomes unsorted, the columns appear in the order they are defined in the database, and all the columns are shown.

Using the Structure view to sort data, and hide and show columns

When working with the Table Editor, the table structure view is available in the Structure tool window or as the corresponding popup.

The structure view shows the list of all the columns and lets you sort the data as well as hide and show the columns.

To open the Structure tool window, do one of the following:

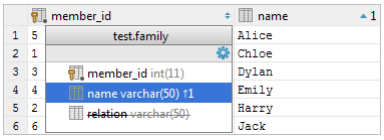
- Select View | Tool Windows | Structure in the main menu.
- Click Structure on the left-hand tool window bar .
- Press `Alt+7`.

To open the structure popup, do one of the following:

- Right-click a cell in the table header row and select Column List.
- Press `Ctrl+F12`.

In the tool window or the popup, select the column of interest and do one of the following:

- To sort the data by this column in the ascending order, press `Shift+Alt+Up`. (In the tool window, you can, alternatively, select Sort | Ascending from the context menu.)
- To sort the data in the descending order, press `Shift+Alt+Down`. (In the tool window, alternatively, Sort | Descending.)
- To cancel sorting by this column, press `Ctrl+Shift+Alt+Backspace`. (In the tool window, alternatively, Sort | Unsorted.)
- To hide the column (or show a hidden column), press `Space`. (The names of hidden columns are shown struck through. In the tool window, alternatively, the Hide Column or Show Column context menu command can be used.)

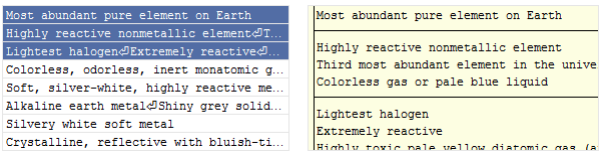


The shortcuts for sorting table data (`Shift+Alt+Up`, `Shift+Alt+Down` and `Ctrl+Shift+Alt+Backspace`) can be used in the Table Editor without opening the structure view.

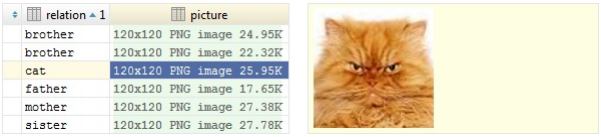
See also, Sorting data, Hiding and showing columns and Restoring the initial table view.

Using the quick documentation view

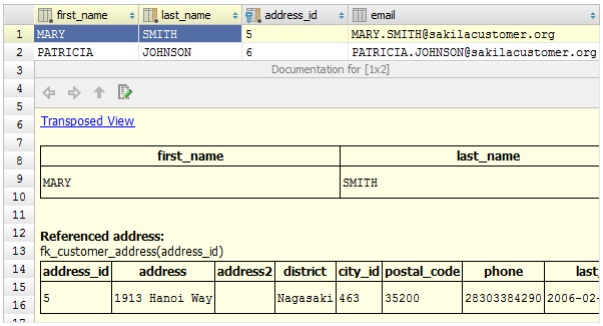
The quick documentation view provides details about the values in the selected cell or cells. For example, if a cell contains long text, normally, you can see only its beginning. The whole text is shown in the quick documentation view.



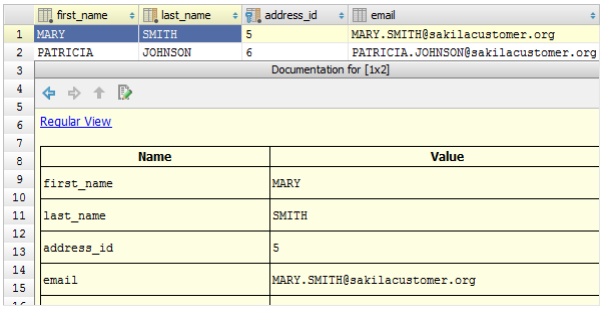
If a cell contains an image, you can see that image in the quick documentation view.



You can also see the records referenced in the current record as well as the records that reference the current one.



If necessary, you can switch to the transposed view. This is when the rows and columns are interchanged. Thus, for a row, the cells are shown one beneath the other.



To open the quick documentation view, press `Ctrl+Q` or select Quick Documentation from the View or the context menu.

To switch to the transposed view, click Transposed View. See also, Transposing the table.

To close the quick documentation view, press `Escape`.

Transposing the table

The transposed table view is available. In this view, the rows and columns are interchanged.

To turn this view on or off, click on the toolbar and select Transpose. Alternatively, use the Transpose context menu command.

Enabling coding assistance for a column

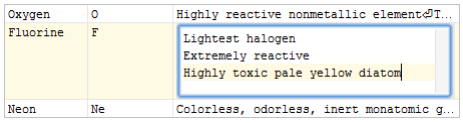
You can assign a column one of the supported languages (e.g. SQL, HTML or XML): right-click the corresponding header cell, select Edit As and select the language. As a result, you get coding assistance for the selected language in all the cells of the corresponding column.

You can also assign a language to an individual cell .

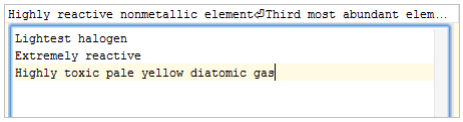
Modifying cell contents

You can modify values in the table cells and, if appropriate, upload files.

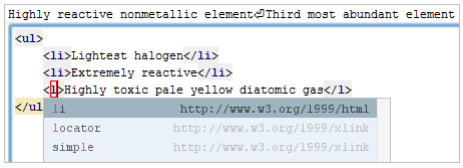
1. To start editing a value or uploading a file, do one of the following:
   - Double-click the corresponding table cell.
   - Right-click the cell and select Edit Value or Edit Maximized from the context menu.
   - Select the cell and press `F2` or `Shift+Enter`. In the latter case, the cell will be maximized.
   - Select the cell and start typing. Note that in this case the initial cell contents are deleted right away and is replaced with the typed value.
2. When in the editing mode, you can:
   - Modify the value right in the cell. To start a new line, use `Ctrl+Enter`. To save the changes, press `Enter`. To restore an initial value and quit the editing mode, press `Escape`.



- Maximize the cell if you need more room for editing. To do that, press `Ctrl+Shift+Alt+M`, or right-click the cell and select Maximize.
  When working in a maximized cell, use `Enter` to start a new line and `Ctrl+Enter` to save the value. To restore an initial value and quit the editing mode, press `Escape`.



- Upload a file into the field (e.g. to replace an existing file with a new one). To do that, right-click the cell and select Load File. Then select the necessary file in the dialog that opens.
- Replace the current value with `null`. To do that, right-click the cell and select Set NULL.
- Edit a value in the cell as a fragment in one of the supported languages (e.g. SQL, HTML or XML). To do that, right-click the cell, select Edit As and select the language. As a result, you get coding assistance for the language you have selected.



Adding a row

If + on the toolbar is enabled, you can add rows to the table.

1. To start adding a row, do one of the following:
   - Click + on the toolbar.
   - Right-click the table and select Add New Row from the context menu.
   - Press `Alt+Insert`.

   Note that the context menu Clone Row command (`Ctrl+D`) can be used as an alternative.
2. Enter the values into the cells. For instructions, see Modifying cell contents.
3. To save the new row, select Submit New Row from the context menu or press `Ctrl+Enter`.

Deleting rows

If – on the toolbar is enabled, you can delete rows. To do that:

1. Select the row or rows that you want to delete.
   Rows are selected by clicking the cells in the column where the row numbers are shown. To select more than one row, use mouse clicks in combination with the `Ctrl` key.
2. Do one of the following:
   - Click – on the toolbar.
   - Press `Ctrl+Y` or `Delete`.
3. Confirm you intention to delete the selected row or rows.

Managing database transactions

The Auto-commit check box, and the Commit and Rollback icons on the toolbar let you manage database transactions.

The Auto-commit check box is used to turn the autocommit mode for the database connection on or off.

In the autocommit mode, each SQL statement is executed in its own transaction that is implicitly committed. Consequently, the SQL statements executed in this mode cannot be rolled back.

(Each change of a value in the table, or adding or deleting a row results in executing an SQL statement.)

If the autocommit mode is off, transactions are committed or rolled back explicitly by means of the Commit ( ) or Rollback ( ) command. Each commit or rollback starts a new transaction which provides grouping for a series of subsequent SQL statements.

In this case, the data manipulations in the transaction scope are committed or rolled back all at once when the transaction is committed or rolled back.

Comparing tables

You can compare the current table with any other table which is open in the table editor or shown in the Database Console tool window. To do that, click on the toolbar and select the table of interest.

The comparison results are shown in the differences viewer .

Copying table data to the clipboard or saving them in a file

1. Do one of the following:
   - Click on the toolbar, and select To File or To Clipboard.
   - Right-click the table, point to Dump Data and select To File or To Clipboard.
2. If you are saving the data in a file, select the output format (the extractor), and then specify the file name and location.

If only a subset of rows is currently shown, all the rows are copied to the clipboard or saved in a file anyway.

See also, Changing data output format.

Copying a selection to the clipboard

To copy selected cells or rows to the clipboard, press `Ctrl+C` or select Copy from the main or the context menu.

If the table is shown transposed (see Transposing the table), the range of cells is copied transposed (as shown) if the Allow Transposition option is on and non-transposed (as in the original table) otherwise.

To turn the Allow Transposition option on or off, right-click the table, point to Data Extractor: <current_format> and click Allow Transposition. Alternatively, click Tab S...(TSV) on the toolbar and click Allow Transposition.

Changing data output format

When you copy table data to the clipboard (the Copy and Dump Data | To Clipboard commands), unless you change the format, the data are output as Tab-Separated Values (TSV).

To change the output format or to define a new format for the output:

1. Do one of the following:
   - Click Tab S...(TSV) on the toolbar.
   - Right-click the table and point to Data Extractor: <current_format>.
2. Do one of the following:
   - To select a different output format, click the name of the format.
   - To define a new delimiter-separated values format, or to view or modify the settings for an existing one, click Configure CSV Formats. As a result, the CSV Formats Dialog will open.

Saving a LOB in a file

If a cell contains a binary large object (a.k.a. BLOB or LOB), you can save such a LOB in a file.

1. Right-click the cell that contains the LOB of interest and select Save LOB To File.
2. In the dialog that opens, specify the name and location of the destination file and click OK.

Updating the table view

To refresh the table view, do one of the following:

- Click on the toolbar.
- Right-click the table and select Reload Page from the context menu.
- Press `Ctrl+F5`.

Use this function to:

- Synchronize the data shown with the actual contents of the database.
- Apply the Result set page size setting after its change.

Viewing the query

To see the query that was used to generate the table:

- Click View Query on the toolbar.
  If necessary, you can select the query text and copy it to the clipboard (`Ctrl+C`).

To close the pane where the query is shown, press `Escape`.

Working with the CREATE TABLE statement

Click the DDL tab (in the lower-left part of the Table Editor) to see the `CREATE TABLE` statement used to create the table.

If necessary, you can edit this statement and then run it (on the toolbar or `Ctrl+Shift+F10`).

To regenerate the `CREATE TABLE` statement for the current state of the table in the database, use or `Ctrl+F5`.

Running SQL Script Files

You can run an SQL file as a whole. You can also execute individual statements contained in an SQL file.

- Running an SQL file
- Executing individual statements

Running an SQL file

When running an SQL script file as a whole:

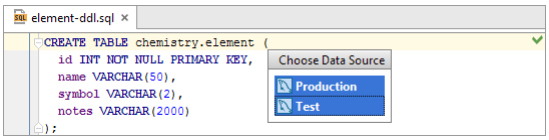
- You don't need to open the file in the editor. You can select the necessary file in the Project tool window.
- You can run the file for more than one data source at once.

On the other hand:

- The statements with parameters won't run.
- Retrieved data for the `SELECT` statements won't be shown.

To run an SQL file:

1. Select the necessary SQL file in the Project tool window, or open the file in the editor.
2. Do one of the following:
   - Select Run "<file_name>" from the context menu.
   - Press `Ctrl+Shift+F10`.
3. In the Choose Data Source pop-up, click the data source to which the script should be applied.
   If you want to run the script for more than one data source, select the data sources of interest in the pop-up and press `Enter`.



Executing individual statements

When running individual statements contained in an SQL file:

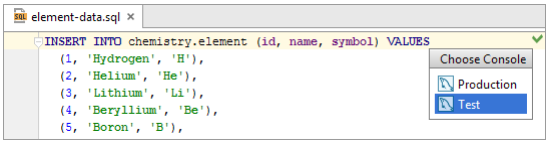
- The statements can contain parameters. Prior to running such statements IntelliJ IDEA will ask you to specify the parameter values.

On the other hand:

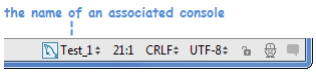
- The statements are run for only one data source at a time.

To run a statement or statements:

1. Open the SQL file of interest in the editor.
2. Place the cursor within the statement you want to execute.
   If you want to run more than one statement, select (highlight) the necessary statements.
3. Do one of the following:
   - Press `Ctrl+Enter` or select Execute from the context menu.
   - Press `Alt+Enter` or click, and select Run query in console.
4. Select the database console to be used.



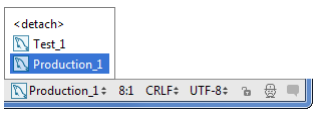
The statement or statements are executed using the selected console. The corresponding console is associated with the file. The name of the associated console is shown on the Status bar.



After that, the Execute and the Run query in console commands start to work differently:

- When using the Execute command, the associated console is used.
- When using the Run query in console command, IntelliJ IDEA asks you to select the console (as before).

You can use the menu on the Status bar to associate the file with a different console.



<detach> in this menu removes the association between the file and the console.

For the Execute command options, see Execute in Console.

Running Injected SQL Statements

You can inject an SQL statement into a string literal (see Using Language Injections) and then run that statement:

1. In the editor, place the cursor within the corresponding string literal.
2. Do one of the following:
   - Press `Ctrl+Enter`.
   - Press `Alt+Enter` and select Run query in console.
   - Click and select Run query in console.
3. If asked, select the database console to be used.
4. If the statement contains parameters, specify the parameter values.

Dart Support

In this section:

- Dart Support
  - Dart support
- Preparing for Dart Development
- Using Integration with the Pub Tool
- Using Integration with the Dart Analysis Server
- Running and Debugging Dart Applications
- Testing Dart

Dart support

IntelliJ IDEA supports developing, running, and debugging Dart code. IntelliJ IDEA recognizes `*.dart` files, and allows you to edit them providing full range of coding assistance. Dart files are marked with the icon.

Dart is intended for developing Web and command-line applications. IntelliJ IDEA supports both workflows.

- For running and debugging Dart command-line applications, IntelliJ IDEA provides a dedicated run/debug configuration of the type Dart Command Line Application.
- To run and debug a Dart Web application in a browser, you either need to use the Dartium browser or have the Dart code translated into JavaScript.
  Most browsers that work with JavaScript code do not natively process Dart code. The only exception is Dartium. This dart-oriented version of Chromium contains a built-in Dart VM and therefore can run Dart code directly. To be executed in any other browser, Dart code has to be compiled into JavaScript.

  With the pub tool, the compilation is performed through the pub serve action using the built-in dart2js compiler.

  You can run Dart Web application in any browser, while debugging is supported only in Dartium and Chrome. To run a Dart Web application you just need to open the starling HTML page in the browser of your choice. Debugging a Dart Web application is initiated through a run configuration of the type JavaScript Debug.

When the Dart development support is enabled, see Preparing for Dart Development, IntelliJ IDEA provides full coding assistance, which involves:

- Code completion for keywords, labels, variables, parameters and functions.
- Error and syntax highlighting.
- Code formatting and folding.
- Numerous code inspections and quick-fixes.
- Integration with the native Dart static analysis engine: all inconsistencies and potential problems are reported in the dedicated Dart Analysis Tool Window with the possibility to navigate to the fragment of the source code where the problem was detected.
- Refactoring:
  - Common refactoring procedures, such as extract method, inline, rename/move, etc.
  - Dart-specific refactoring procedures, such as change signature, extract parameter, extract variable.

  See JavaScript-Specific Refactorings for details.
- Code generation:
  - Generating code stubs based on file templates during file creation.
  - Ability to create line and block comments (`Ctrl+Slash` / `Ctrl+Shift+Slash`).
- Navigation and search through the source code:
  - Navigating with Structure View.
  - Navigate | Declaration (`Ctrl+B`).
  - Navigate | Implementation (`Ctrl+Alt+B`) from overridden method / subclassed class.

Preparing for Dart Development

On this page:

- Before you start
- Downloading the Dart Tools
- Configuring Dart Support in a Project
  - Creating a Dart Application Stub
  - Configuring Dart Support in an Existing Project
  - Developing Several Dart Projects (Packages) within one IntelliJ IDEA Project

Before you start

Install and enable the Dart plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Downloading the Dart Tools

To have all the tools for developing both server-side and client-side applications, download the Dart SDK and the Dartium browser in which you can run Dart code without previously transpiling it into JavaScript. The Dart SDK incorporates the Dart Virtual Machine, the Dart Libraries, and all the background command line tools, the most important of them being the dart2js transpiler and the Dart-specific pub package manager .

Learn more about Dart development tools at the Dart Download Page .

Configuring Dart Support in a Project

To develop a Dart application you need to tell IntelliJ IDEA where the Dart SDK and the Dartium executable file are located. If you are using the Dart project template, IntelliJ IDEA will ask you to specify these settings during the project creation. Alternatively you can specify these paths at any time later on the Dart page of the Settings dialog box.

These settings are specific for each project. This means that you can have several Dart SDKs and Dartium installations in IntelliJ IDEA and switch between them from one project to another.

Next, you need a project with Dart-specific structure. If you are using the Dart project template, IntelliJ IDEA establishes the required folder structure for you automatically. Alternatively, you can add and arrange the necessary folders manually.

Creating a Dart Application Stub

During project creation, IntelliJ IDEA can generate a stub for developing Dart applications. The project structure is set up and some sources are generated based on the external Dart templates downloaded upon your request.

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Dart in the left-hand pane.
3. In the right-hand pane, specify the paths to the Dart SDK and to the Dartium executable file. IntelliJ IDEA detects and displays the Dart version. If the SDK was downloaded within the Dart Editor Bundle, the path to Dartium is also detected and filled in automatically. Click Next.
4. On the second page of the wizard,
   1. In the Location text box, specify the path to the project folder where the project-related files will be stored.
   2. Specify the paths to the Dart SDK and to the Dartium executable file. IntelliJ IDEA detects and displays the Dart version. If the SDK was downloaded within the Dart Editor Bundle, the path to Dartium is also detected and filled in automatically.
   3. To have annotations verified and assertions checked, select the Start Dartium in checked mode check box. Learn more about Dartium command line flags at https://www.dartlang.org/tools/dartium/ .

   Click Finish when ready.

If necessary, you can open the Dart page of the Settings dialog box by clicking the Settings button and configure command-line options and a custom user data directory for Dartium.

Configuring Dart Support in an Existing Project

Besides creating a Dart project from the very beginning, you can configure Dart support in a project at any stage of its development.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Dart under Languages & Frameworks.
2. Select the Enable Dart support in project <project name> check box whereupon all the fields below become available.
3. In the Dart SDK Path text box, specify the location of the downloaded Dart SDK. Type the path manually or click the Browse button and choose the path in the dialog box that opens.
   If IntelliJ IDEA recognizes the Dart SDK correctly, it displays the revision number of the detected SDK in the Version read-only field.
4. In the Dartium Path text box, specify the location of the Dartium executable file. In the Dartium browser , you can run Dart code without previously transpiling it into JavaScript. Type the path manually or click the Browse button and choose the path in the dialog box that opens. If the SDK was downloaded within the Dart Editor Bundle, the path to Dartium is also detected and filled in automatically. Learn more about Dart development tools at the Dart Download Page .
5. To have annotations verified and assertions checked, select the Start Dartium in checked mode check box. Learn more about Dartium command line flags at https://www.dartlang.org/tools/dartium/ .
6. In the Enable Dart support for the following modules area, select the check boxes next to the names of the modules where you need Dart support. If you want to use a custom Dart package in a module, specify the location of the packages in the Custom package roots field.
   IntelliJ IDEA detects the specified Dart SDK and adds it as module dependencies, see Configuring Module Dependencies and Libraries .

Developing Several Dart Projects (Packages) within one IntelliJ IDEA Project

If your application consists of several Dart projects (packages), you can develop all of them within one single IntelliJ IDEA project. To attach a Dart project (package) to an existing IntelliJ IDEA project you need to add its root folder as a content root.

To add a Dart project (package) to an existing IntelliJ IDEA project:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Directories under Project: <project name>.
2. In the right-hand pane which shows a list of already existing content roots, click Add Content Root. Then in the dialog box that opens, choose the root folder of the Dart project (package) to be attached.

Using Integration with the Pub Tool

On this page:

- Introduction
- Managing Dart Dependencies
- Compiling Dart Code into JavaScript using the Pub Serve Tool
- Building a Dart Application

Introduction

IntelliJ IDEA provides interface for using the Pub tool . Alternatively, you can use the tool in the command line mode from the embedded local terminal . The pub tool is intended for installing, uninstalling, and upgrading Dart packages for your project, compiling Dart code into JavaScript for further running or debugging, and for building Dart applications.

The pub tool is included in the Dart SDK, so when you download the SDK you automatically get the pub tool. For details, see Preparing for Dart Development and Dart Download Page .

You can also use the pub tool in the command line mode from the embedded local terminal .

Execution of the pub tool commands is logged in the Messages Tool Window . In this tool window, you can examine the tool log and re-run the last executed command by clicking on the toolbar.

Managing Dart Dependencies

IntelliJ IDEA provides interface for running the pub get and pub upgrade actions from the editor. As soon as you open a pubspec.yaml file in the editor, IntelliJ IDEA displays a Pub actions pane at the top of the `pubspec.yaml` editor tab. You can use the links on this pane or use the context menu of the editor or the `pubspec.yaml` file in the Project tool window.

The pub tool saves the downloaded packages in caches, creates a `packages` folder next to the `pubspec.yaml` file which contains information on packages and their dependencies. In the `web` folder, the pub tool creates a copy of the `packages` folder. As soon as a subfolder if created under the `web` folder, a copy of the `packages` folder is created in the new subfolder.

All the copies of the `packages` folder are automatically excluded from the project, that is, they are not indexed. IntelliJ IDEA excludes these copies on opening a project and every time the pub get or pub upgrade action is invoked. All the excluded copies of the `packages` folder are listed in the exclusion list on the Directories page. You can remove a copy from this list but the copy will be restored during the first automatic exclusion.

To download or upgrade dependencies:

1. Open the `pubspec.yaml` file in the editor.
2. To detect and download dependencies, do one of the following:
   - Click Get Dependencies on the Pub actions pane.
   - Choose Pub: Get Dependencies on the context menu of the editor.
   - In the Project tool window, select the `pubspec.yaml` file and choose Pub: Get Dependencies on the context menu of the selection.
3. To download the up-to-date versions of the existing dependencies, do one of the following:
   - Click Upgrade Dependencies on the Pub actions pane.
   - Choose Pub: Upgrade Dependencies on the context menu of the editor.
   - In the Project tool window, select the `pubspec.yaml` file and choose Pub: Upgrade Dependencies on the context menu of the selection.

If for some reasons the previously downloaded and stored in a cache dependencies are broken, you can clear the cache and download the dependencies anew:

1. Open the `pubspec.yaml` file in the editor.
2. Click Repair Cache on the Pub actions pane.

Compiling Dart Code into JavaScript using the Pub Serve Tool

When running and debugging Dart web applications, IntelliJ IDEA uses the Pub Serve tool. Integration with it is established through the built-in server which acts transparently as a proxy server between your application and the Pub Serve tool. In other words, integration with the Pub Serve tool works invisibly and you do not need to care about it. The tool is invoked automatically when you run or debug a Dart web application, either by opening an HTML file or by launching a run/debug configuration. The tool analyzes the `<script></script>` element in the HTML file. If the element references a Dart file, Pub Serve invokes the Dart2js compiler whereupon the compiled JavaScript files are passed to the browser where they are processed.

The work of the Pub Serve tool is logged in the dedicated Pub Serve tool window which opens when you start running or debugging a Dart web application for the first time during the current IntelliJ IDEA session. You can stop the tool by clicking on the toolbar. As soon as you start running or debugging again, the tool restarts automatically.

See Running and Debugging Dart Web Applications for details.

Building a Dart Application

When you are ready to deploy your application, you need to build it, which includes generating assets for the project and for all its dependencies and compiling Dart code into JavaScript. Compilation is necessary because

most browsers that work with JavaScript code do not natively process Dart code. The only exception is Dartium . This dart-oriented version of Chromium contains a built-in Dart VM and therefore can run Dart code directly. To be executed in any other browser, Dart code has to be compiled into JavaScript.

With the pub tool, the compilation is performed through the pub serve action using the built-in dart2js compiler.

IntelliJ IDEA provides interface for using the pub tool which builds applications through the pub build action. The pub build action compiles the Dart code into JavaScript using the dart2js , generates assets , and stores all the output in the automatically created `build` folder. The folder is marked as excluded, see Configuring Folders Within a Content Root .

See Assets and Transformers for details.

To build a Dart application:

1. Open the `pubspec.yaml` file in the editor or switch to the tab where it is opened.
2. On the pub pane at the top of the tab, click Build.
3. In the dialog box that opens, choose the build mode which determines the behaviour of the dart2js tool:
   - Release: choose this option to have dart2js minify the generated JavaScript code. The term minification or compression in the context of JavaScript means removing all unnecessary characters, such as spaces, new lines, comments without affecting the functionality of the source code. In the Release mode, the source `.dart` files are not included in the build output. This mode is applicable when you are going to publish your application.
   - Debug: choose this option if you do not want to compress the generated JavaScript. In this mode, the generated JavaScript files are included in the build output without compression, as well as the source `.dart` files.
   - Other: choose this option if you want to use a custom transformer and specify the path to it.

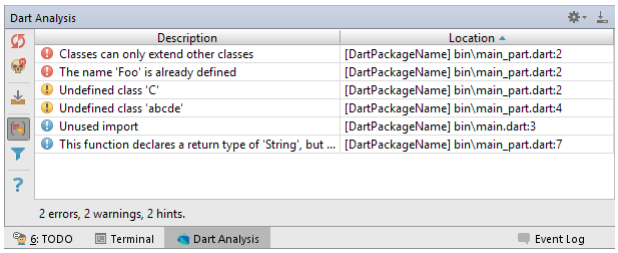
Using Integration with the Dart Analysis Server

On this page:

- Introduction
- Messages lst
- Grouping and sorting
- Navigation and speed search
- Configuring the scope of files to view problem reports from

Introduction

IntelliJ IDEA provides interface for using the Dart Analyzer tool . The tool is intended for static analysis of your Dart source code. All inconsistencies and potential problems are reported in the dedicated Dart Analysis tool window with the possibility to navigate to the fragment of the source code where the problem was detected.



Messages lst

All the problem reports are displayed in a table which consists of two columns, Description and Location, and a summary below.

- The Description column shows the message itself, which can be an error message, a warning, or a hint.
- The Location column shows the following data:
  - The name of the Dart package in which the problem arose. The Dart package name corresponds to the `project name` from `pubspec.yaml` .
  - The relative from the the Dart package root to the file where the problem arose.
  - The line number where the problem arose.
- The Summary area below the table shows the number of detected errors, warnings, or hints and the filter status.

Grouping and sorting

The messages in the Dart Analysis tool window can be grouped and sorted. By default, the messages are primarily sorted by their severity, that is, the error messages are shown at the top of the list, then come warnings, and finally hints are displayed. In each severity group, messages are grouped by the Dart package name, which is super handy for projects with multiple `pubspec.yaml` files. In each package group, the messages are grouped by their file paths. Finally, in each file group, the problems are sorted by the line number where they occurred.

You can re-configure the secondary sorting:

- Click the Location column header to switch between the ascending and descending order of the Dart package name + file path sorting. Note that this does not affect the order of severity groups, that is, errors are always shown first. Line numbers are not taken into consideration either, for each particular file problems are always shown in ascending order of line numbers.
- Click the Description column header to perform the secondary sorting alphabetically by the problem description. Note that this does no affect the primary grouping by severity.

To disable the primary grouping by severity, release the Group by Severity toggle button on the toolbar. After that the severity of a problem is not taken into consideration at all and problem reports are sorted only by their Description or Location (Dart package name + file path). This sorting method guarantees that all the problems for each particular file are grouped together in the table, regardless of their severity.

Navigation and speed search

IntelliJ IDEA provides speed search through the messages in the Dart Analysis tol window with the possibility to navigate from a problem report to the fragment of the source code where the problem was detected.

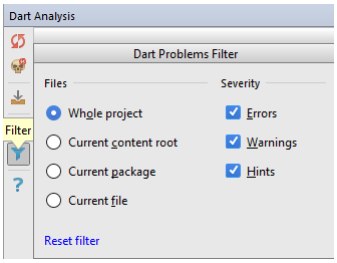
- To locate problems for a specific file or to find a particular message, move the focus to the table and start typing the file name or the text of the message. IntelliJ IDEA detects the matching row and selects it.
  Fuzzy search with partial and middle matching is also supported. For example, if you start typing `ain` , the messages related to the `main.dart` fill be also detected.
- To navigate from a problem report to the source code where the problem occurred, select the message in question and choose Jump to Source on the context menu of the selection.
- To have IntelliJ IDEA automatically open the source file with the problem reported in the selected row, press the Autoscroll to Source toggle button on the toolbar.

Configuring the scope of files to view problem reports from

You can configure the criteria according to which a problem report is displayed in the tool window or not:

1. Click the Filter button on the toolbar.
2. In the Dart Problems Filter pop-up window, specify the following:
   - In the Severity area, specify the types of messages to be shown, the available types are Errors, Warnings, and Hints. To have the problems of a severity level displayed, select the check box next to this severity level.
   - In the Files area, choose the scope for which you want to see problem reports. The available options are:
     - Whole project
     - Current content root
     - Current package: when this option is selected but no `pubspec.yaml` file is detected up the folder hierarchy starting from the current file, then filtering is performed according to the content root where current file is located.
     - Current file

- Click the Reset all filters link to restore the default filter settings.



Filtering is applied immediately as soon as you change the current settings.

Running and Debugging Dart Applications

Dart is intended for developing Web and command-line applications. IntelliJ IDEA supports both workflows.

- For running and debugging Dart command-line applications, IntelliJ IDEA provides a dedicated run/debug configuration of the type Dart Command Line Application.
- To run and debug a Dart Web application in a browser, you either need to use the Dartium browser or have the Dart code translated into JavaScript.
  Most browsers that work with JavaScript code do not natively process Dart code. The only exception is Dartium . This dart-oriented version of Chromium contains a built-in Dart VM and therefore can run Dart code directly. To be executed in any other browser, Dart code has to be compiled into JavaScript.

  With the pub tool, the compilation is performed through the pub serve action using the built-in dart2js compiler.

  You can run Dart Web application in any browser, while debugging is supported only in Dartium and Chrome. To run a Dart Web application you just need to open the starling HTML page in the browser of your choice. Debugging a Dart Web application is initiated through a run configuration of the type JavaScript Debug.

In this section:

- Running and Debugging Dart Command-Line Applications
- Running and Debugging Dart Web Applications

Running and Debugging Dart Command-Line Applications

For running and debugging Dart command-line applications, IntelliJ IDEA provides a dedicated run/debug configuration of the type Dart Command Line Application.

On this page:

– Running a Dart Command-Line Application
– Debugging a Dart Command-Line Application

Running a Dart Command-Line Application

1. Create a run configuration:
   1. On the main menu, choose Run | Edit Configurations.
   2. Click the Add New Configuration toolbar button, and choose Dart Command Line Application on the context menu.
   3. In the Run/Debug Configuration: Dart Command Line Application dialog box that opens, specify the Dart file to start the application from. The file must contain a `main()` method. Optionally specify the options to launch the Dart Virtual Machine with and the program options to start the application with.
   4. Click OK to save the configuration settings.
2. Choose the newly created configuration in the Select run/debug configuration drop-down list on the tool bar and click the Run toolbar button.
3. View the results in the Console tab of the Run tool window.

Debugging a Dart Command-Line Application

With IntelliJ IDEA, you can debug Dart command-line applications in two modes:

– Local debugging. In this mode, you launch the application to debug and start a debugging session through one same run configuration.
– Remote debugging. In this mode, you initiate a debugging session and connect to an already running application. This application can be launched on another host machine or you can previously start it on your machine through another run configuration.

To debug a Dart command-line application locally:

1. Configure and set breakpoints in the Dart code.
2. Create a run configuration:
   1. On the main menu, choose Run | Edit Configurations.
   2. Click the Add New Configuration toolbar button, and choose Dart Command Line Application on the context menu.
   3. In the Run/Debug Configuration: Dart Command Line Application dialog box that opens, specify the Dart file to start the application from. The file must contain a `main()` method. Optionally specify the options to launch the Dart Virtual Machine with and the program options to start the application with.
   4. Click OK to save the configuration settings.
3. Choose the newly created configuration in the Select run/debug configuration drop-down list on the tool bar and click the Debug toolbar button.
4. In the Debug Tool Window that opens, step through the program , stop and resume program execution, examine it when suspended , etc.

To debug a Dart command-line application remotely:

1. Create a Dart Remote Debug run configuration:
   1. On the main menu, choose Run | Edit Configurations.
   2. Click the Add New Configuration toolbar button, and choose Dart Command Line Application on the context menu.
   3. In the Run/Debug Configuration: Dart Remote Debug dialog box that opens, specify the host where the Dart application is running and the port for the debugger to communicate with this host. The specified port is shown in the Command line arguments for the remote Dart VM read-only field. Note that the application must be started exactly with these arguments, so do not forget to provide this information to the one who will start it on the remote host or specify the arguments in the Dart Remote Debug run configuration if you are going to launch the application yourself on the localhost.
   4. From the Search Sources in drop-down list, choose the Dart project to debug if your IntelliJ IDEA project contains several Dart projects configured as content roots.
   5. Click OK to save the configuration settings.
2. Make sure the command-line application to debug is running with the arguments from the Command line arguments for the remote Dart VM read-only field in the Dart Remote Debug run configuration.
3. Choose the newly created Dart Remote Debug configuration in the Select run/debug configuration drop-down list on the tool bar and click the Debug toolbar button.
4. In the Debug Tool Window that opens, step through the program , stop and resume program execution, examine it when suspended , etc.

Running and Debugging Dart Web Applications

On this page:

- Introduction
- Running a Dart Web Application
- Debugging a Dart Web Application

Introduction

To run and debug a Dart Web application in a browser, you either need to use the Dartium browser or have the Dart code translated into JavaScript.

most browsers that work with JavaScript code do not natively process Dart code. The only exception is Dartium . This dart-oriented version of Chromium contains a built-in Dart VM and therefore can run Dart code directly. To be executed in any other browser, Dart code has to be compiled into JavaScript.

With the pub tool, the compilation is performed through the pub serve action using the built-in dart2js compiler.

You can run Dart Web application in any browser, while debugging is supported only in Dartium and Chrome. To run a Dart Web application you just need to open the starling HTML page in the browser of your choice. Debugging a Dart Web application is initiated through a run configuration of the type JavaScript Debug.

When running and debugging Dart web applications, IntelliJ IDEA uses the Pub Serve tool. Integration with it is established through the built-in server which acts transparently as a proxy server between your application and the Pub Serve tool. In other words, integration with the Pub Serve tool works invisibly and you do not need to care about it. The tool is invoked automatically when you run or debug a Dart web application, either by opening an HTML file or by launching a run/debug configuration. The tool analyzes the `<script></script>` element in the HTML file. If the element references a Dart file, Pub Serve invokes the Dart2js compiler whereupon the compiled JavaScript files are passed to the browser where they are processed.

The work of the Pub Serve tool is logged in the dedicated Pub Serve tool window which opens when you start running or debugging a Dart web application for the first time during the current IntelliJ IDEA session. You can stop the tool by clicking on the toolbar. As soon as you start running or debugging again, the tool restarts automatically.

Running a Dart Web Application

No run configuration is required for launching Dart Web applications.

1. In the editor, open the HTML file with the Dart reference. This HTML file does not necessarily have to be the one that implements the starting page of the application.
2. Do one of the following:
   - Choose View | Preview file in on the main menu or press `Alt+F2`. Then select the desired browser from the pop-up menu.
   - Hover your mouse pointer over the code to show the browser icons bar: Click the icon that indicates the desired browser.

   If you choose Dartium which has a built-in Dart virtual machine, simply the Dart code is executed. If you choose another browser, the Dart code is compiled into JavaScript through the Dart2js tool, which is a part of the Pub tool. Whether such compilation is required or not is decided by the Pub Serve tool.

Debugging a Dart Web Application

Debugging of Dart Web applications is supported only in Dartium and Chrome through the JetBrains Chrome Extension. A debugging session is initiated through a run configuration of the type JavaScript Debug.

1. Install and configure the Chrome Extension in your browser:
   1. Open the Chrome Web Store at https://chrome.google.com/webstore/detail/jetbrains-ide-support/hmhgeddbohgjknpmjagkdomcpobmllji .
   2. On the JetBrains IDE Support page that opens, click the Add to Chrome button + FREE . Then click Add in the Confirm New Extension dialog box that opens. Learn more at Using JetBrains Chrome Extension , chapter Installing the JetBrains Chrome Extension.
   3. Make sure the extension connects to IntelliJ IDEA through the same port as the one specified in the Built-in server port field on the Debugger page of the Settings dialog box. To do that, right-click the JB icon, choose Options on the context menu, and then check the value in the Port spin box. Learn more at Using JetBrains Chrome Extension , chapter Changing the Port for Connecting to IntelliJ IDEA.
2. Configure and set breakpoints in the Dart code.
3. Create a run/debug configuration of the type JavaScript Debug :
   1. On the main menu, choose Run | Edit Configurations.
   2. Click the Add New Configuration toolbar button +, and choose JavaScript Debug on the context menu.
   3. In the Run/Debug Configuration:JavaScript Debug dialog box that opens, choose the browser to debug the application in. If you choose Dartium which has a built-in Dart virtual machine, simply the Dart code is executed. If you choose Chrome, the Dart code is compiled into JavaScript through the Dart2js tool, which is a part of the Pub tool. Whether such compilation is required or not is decided by the Pub Serve tool.
      Specify the URL address of the HTML file the references the Dart code to debug. Make sure that the port specified in this URL address is the same as the Built-in server port on the Debugger page of the Settings dialog box and the port specified in the settings for the Chrome Extension.
4. Initiate a debugging session: choose the created run configuration from the Edit configurations drop-down list on the toolbar and click . IntelliJ IDEA opens the specified URL in the chosen browser and opens the Debug Tool Window .
5. In the Debug tool window, step through the program , stop and resume the program, examine it when suspended, etc. See Stepping Through the Program , Pausing and Resuming the Debugger Session , and Examining Suspended Program for details. ,

Testing Dart

On this page:

- Before you start
- Creating a Dart test run configuration

Before you start

1. Make sure the Dart development environment is configured, see Preparing for Dart Development .
2. Make sure the Dart plugin is activated. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Creating a Dart test run configuration

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button +, and choose Dart Test on the context menu.
3. In the Run/Debug Configuration: Dart Test dialog that opens, specify the name of the configuration and the scope of the tests to execute. The available options are:
   - All in File: choose this option to to have IntelliJ IDEA run all the test methods in a file and specify this file in the Test File text box.
   - All in Folder: choose this option to to have IntelliJ IDEA run all the test methods from the files stored in a folder and specify this folder in the Test Folder text box.
   - Group or test by name: choose this option to to have IntelliJ IDEA run a specific tast or a group of tests.
     - In the Test File text box, specify the file where the test or the group is defined.
     - In the Group or Test Name text box, type the name of the test or the group of tests to execute.

     See more about grouping tests at https://www.dartlang.org/articles/dart-unit-tests/#grouping-tests .

Docker

You can add Docker support to IntelliJ IDEA by installing the Docker integration plugin .

- Prerequisites
- Overview of Docker support
- Working with Docker: Process overview
- Example: Deploying a web app into the Wildfly Docker container
- Example: Using Docker Compose

Prerequisites

Make sure that the following prerequisites are met:

- Docker is installed, as described on the page Docker Docs . You can install Docker on the various platforms:
  - Windows
  - Mac OS X
  - Linux (Ubuntu, other distributions-related instructions are available as well);
  - Cloud platforms (e.g. here are tutorials on installing Docker on Amazon EC2 , Google Cloud Platform , and Microsoft Azure .)
- Plugin Docker integration is installed.

  The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Overview of Docker support

The Docker integration plugin adds the following to IntelliJ IDEA:

- Docker configurations. These are named sets of settings for accessing the Docker Remote API.
- Docker Deployment run/debug configurations . They let you download and build Docker images, create and start Docker containers, and deploy your application artifacts into the containers. Application debugging is supported only for Java.
- Docker tool window that lets you manage your Docker images and containers.
- Docker Registry configurations that represent your Docker image repository user accounts.

Working with Docker: Process overview

1. Install Docker and start the Docker daemon (on Windows and OS X - along with the Docker VM in which it runs). For corresponding instructions, see Get Started with Docker .
2. Install the Docker integration plugin. For instructions, see Installing, Updating and Uninstalling Repository Plugins .
   Note: For the plugin to work properly, IntelliJ IDEA should be started using Java 7 or a later version.
3. Create a Docker configuration. You can do that:
   - Separately, in the Settings / Preferences dialog: `Ctrl+Alt+S` | Build, Execution, Deployment | Clouds | + | Docker.
   - When creating a Docker Deployment run configuration: Run | Edit Configurations | + | Docker Deployment, etc.

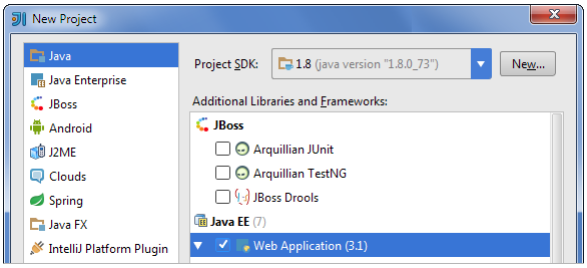
   See the settings for a Docker configuration .
4. If you are going to deploy your application into a container, create an artifact configuration for your application and build the artifact (e.g. Build | Build Artifacts).
5. Create a Docker Deployment run configuration .
6. Execute the run configuration.
7. Use the Docker Tool Window to manage your Docker images, containers and deployed applications.

Example: Deploying a web app into the Wildfly Docker container

Once you have installed Docker and the Docker integration plugin, you can try out some of the Docker integration features.

In this example, we'll use the Wildfly app server Docker image to create a container and deploy a one-page JSP application into that container.

1. Create a project for developing a Java web application (File | New | Project | Java | Web Application, etc.), see e.g Enabling Web app development support when creating a project or module.

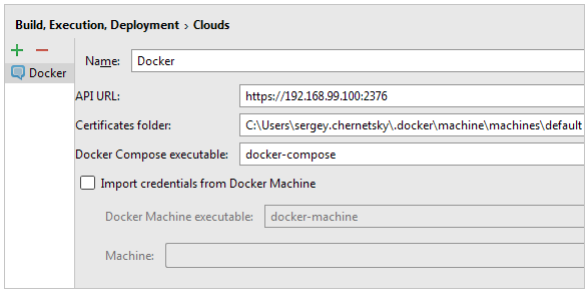


2. When the project is created, add text (e.g. `Hello World!`) to `index.jsp` , see e.g. Developing source code.

3. Create a Docker configuration and specify Docker API settings:

1. Open the Clouds page of the Settings / Preferences dialog: File | Settings | Build, Execution, Deployment | Clouds on Windows and Linux, or IntelliJ IDEA | Preferences | Build, Execution, Deployment | Clouds on OS X.
2. Click + and select Docker.
   The default settings provided by IntelliJ IDEA should be OK:
   - API URL: `https://192.168.99.100:2376`
   - Certificates folder: `<your_home_directory>\.docker\machine\machines\default`

   The rest of the settings, at the moment, don't matter.



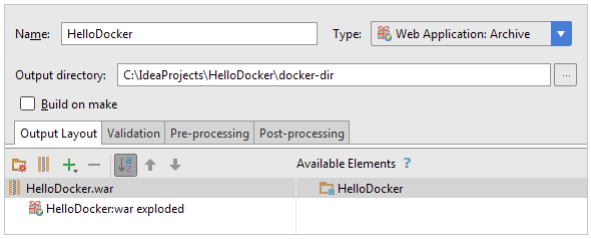
3. Click OK in the Settings / Preferences dialog.

4. In the project root directory, create a new directory (e.g. `docker-dir`): File | New | Directory, etc.
   We'll use this directory to store our `Dockerfile`, a `.war` application artifact and a container settings file - just to keep all the files for working with Docker in one place.
5. Create a file `Dockerfile` in the `docker-dir` directory.
6. Add the following to your Dockerfile:

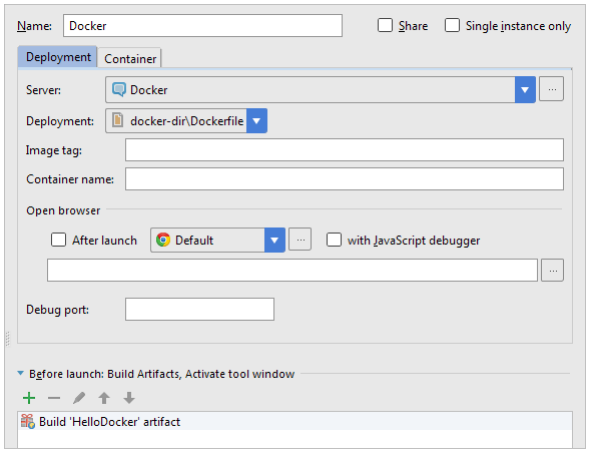
```
FROM jboss/wildfly
                    ADD <artifact-name>.war /opt/jboss/wildfly/standalone/deployments/
```

   Use the actual artifact name in place of <artifact-name>. On the following picture, the name of the artifact is `HelloDocker`.

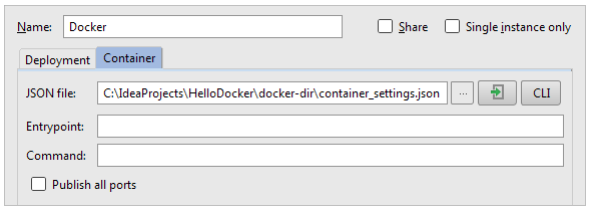
   (This Dockerfile sets `jboss/wildfly` as the base image and copies the local file `<artifact-name>.war` located in the same directory as the Dockerfile to the container directory `/opt/jboss/wildfly/standalone/deployments/`.)
7. Create an artifact configuration for your <artifact-name>.war:
   1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`) and select Artifacts.
   2. Click +, select Web Application: Archive and select For '<project-name>:war exploded'.
   3. Change the artifact name (in the Name field). The name should be the same as in your `Dockerfile` (`<artifact-name>`) but without `.war` at the end.
   4. Select the `docker-dir` directory as the artifact output directory (the Output directory field).



   5. Click OK in the Project Structure dialog.
8. Create a Docker Deployment run/debug configuration and specify its settings:
   1. Run | Edit Configurations | + | Docker Deployment.
   2. Name. Specify the run configuration name, e.g. Docker.
   3. Server. Select your Docker configuration.
   4. Deployment. Select docker-dir/Dockerfile. This means that the application deployment will be controlled by the corresponding `Dockerfile`.
   5. Before launch. Click +, select Build Artifacts, and select your `<artifact-name>` artifact.



6. Select the Container tab.
7. JSON file. Generate a sample container settings `.json` file and save it in `docker-dir`: click, select `docker-dir` and click OK.



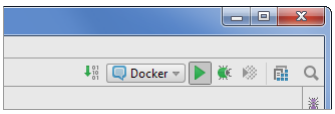
The rest of the settings are optional.

8. Click OK in the Run/Debug Configurations dialog.
9. Open `container-settings.json` in the editor. Note this fragment:

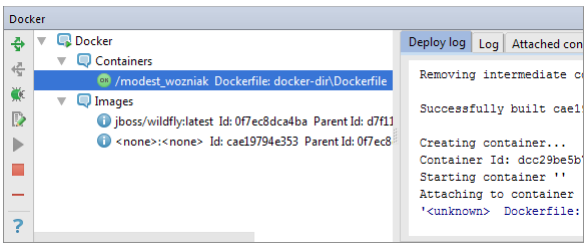
```
"HostConfig": {
                    "PortBindings":{ "8080/tcp": [{ "HostIp": "0.0.0.0", "HostPort":
                    "18080" }] }
                    }
```

This means that the TCP port 8080 inside the container will be available to the outside world as the port 18080.

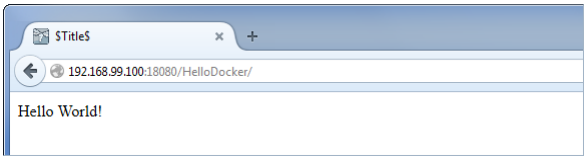
0. Execute the run/debug configuration, e.g. by clicking to the right of the run configuration selector.



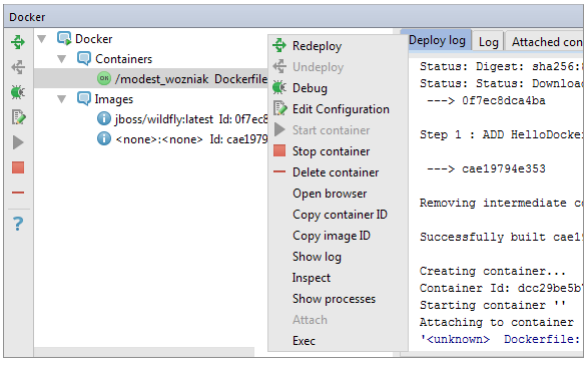
Wait while IntelliJ IDEA downloads and builds the Wildfly image, deploys the application artifact, and creates and starts the container. (The Docker tool window opens and corresponding info is output there.)



1. When the container is finally started, open a web browser and go to `http://192.168.99.100:18080/<artifact-name>/` to see the application output.



2. To get an impression of what other functionality is available, look at context menu commands for various items in the Docker tool window.



Note that you can also run Bash `docker` commands. On OS X and Linux you can use the IntelliJ IDEA Terminal tool window for that. On Windows you should use the Bash command-line interpreter available in the Docker Toolbox.

Example: Using Docker Compose

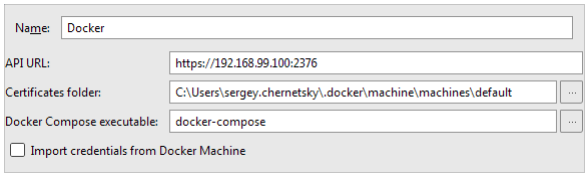
The project created in the previous example is used for illustrating the workflow.

To use Docker Compose , you should:

1. Make sure that the Docker Compose executable setting in your Docker configuration is correct.
   1. On Windows or Linux: File | Settings | Build, Execution, Deployment | Clouds.

      On OS X: IntelliJ IDEA | Preferences | Build, Execution, Deployment | Clouds.
   2. If there is more than one configuration on the Clouds page, select the Docker configuration of interest under + and –.
   3. The default setting `docker-compose` for Docker Compose executable is fine if:
      - The actual name of the executable file is `docker-compose` .
      - The path to the directory where the file is located is included in the environment variable `Path` .

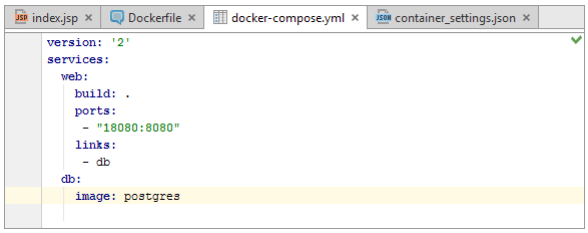
      If necessary, you can specify an actual path to the executable file: click ... and select the file in the dialog that opens.

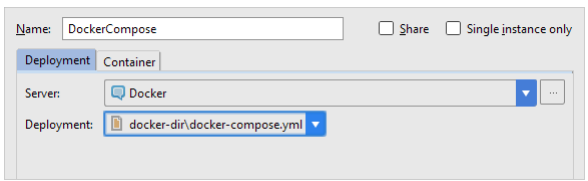
   4. Click OK in the Settings / Preferences dialog.
2. Create a Docker Compose YAML file and define your services.
   1. Create the file `docker-compose.yml` in the `docker-dir` directory.
   2. Copy the following service definitions into the file:

      ```
      version: '2'
                          services:
                          web:
                          build: .
                          ports:
                          - "18080:8080"
                          links:
                          - db
                          db:
                          image: postgres
      ```

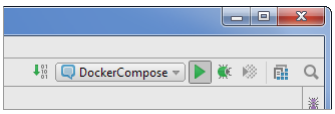
      (This file defines two services: `web` and `db` . The container for the `web` service is built according to the `Dockerfile` located in the same directory. The container port `8080` is mapped to the host port `18080` . The service is linked to a container in the `db` service. The container for the `db` service is built using the `postgres` image.)

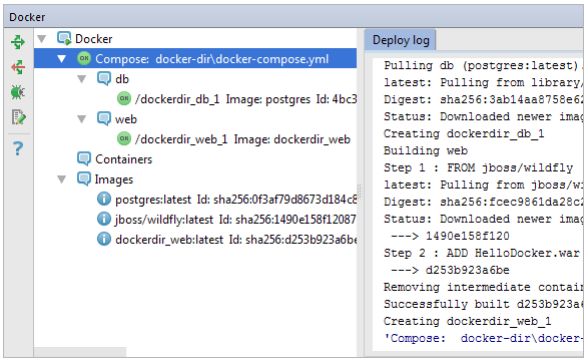
3. Create a Docker Deployment run configuration. In that configuration, specify that the YAML file should be used to deploy your multi-container application.
   1. Run | Edit Configurations | + | Docker Deployment.
   2. Name. Specify the run configuration name, e.g. DockerCompose.
   3. Server. Select your Docker configuration.
   4. Deployment. Select docker-dir/docker-compose.yml. This means that the application deployment will be controlled by the corresponding Docker Compose YAML file.

      The rest of the settings are optional.
   5. Click OK in the Run/Debug Configurations dialog.
4. Execute the run configuration (▶).

   The result in the Docker tool window will finally look something like this:


Docker
Docker
Compose: docker-dir\docker-compose.yml
db
/dockerdir_db_1 Image: postgres Id: 4bc3
web
/dockerdir_web_1 Image: dockerdir_web
Containers
Images
postgres:latest Id: sha256:0f3af79d8673d184c8
jboss/wildfly:latest Id: sha256:1490e158f12087
dockerdir_web:latest Id: sha256:d253b923a6be
Deploy log
Pulling db (postgres:latest)
latest: Pulling from library/
Digest: sha256:3ab14aa8758e62
Status: Downloaded newer imag
Creating dockerdir_db_1
Building web
Step 1 : FROM jboss/wildfly
latest: Pulling from jboss/wi
Digest: sha256:fcec9861da28c2
Status: Downloaded newer imag
---> 1490e158f120
Step 2 : ADD HelloDocker.war
---> d253b923a6be
Removing intermediate contai
Successfully built d253b923a6
Creating dockerdir_web_1
'Compose: docker-dir\docker-

EJB

In this section:

- EJB
  - General EJB features
  - EJB 3.0-specific features
- Enabling EJB Support
- Creating EJB
- Configuring Primary Key
- Configuring Service Endpoint
- Creating and Editing Assembly Descriptors
- Creating CMP Bean Fields
- Creating Local and Remote Interfaces
- Creating Message Listeners
- Creating Transfer Objects
- Defining Bean Class and Package
- Editing Module with EJB Facet
- Migrating to EJB 3.0
- Using EJB ER Diagram

General EJB features

IntelliJ IDEA features the complete EJB support. It understands EJB specifications from 1.x to 3.0, and leverages them through all of its productivity-boosting features - from coding assistance to refactoring, creating the environment ideally suited for developing EJB applications.

- File and live templates for entity, session, message and other beans.
- Auto-generating code for CMP beans, fields and relationships.
- Dedicated context editors for all supported beans.
- Automatic building of standard EJB deployment packages.
- Automatic generation of appropriate EJB XML descriptors.
- Complete EJB-aware coding assistance:
  - Code completion for both EJB code and descriptor files.
  - EJB error highlighting.
  - EJB-aware intention actions and quick fixes .
  - Powerful EJB-aware refactorings .

IntelliJ IDEA unifies all EJB-related application parts in the dedicated Facet , which includes EJB descriptors, build, and library settings.

Multiple EJB facets are allowed per module.

EJB 3.0-specific features

- Annotation mechanism for creating EJB, and Interceptors.
- Automatic code generation, completion and dedicated binding editor.
- Full EJB persistence support, powered with generating of persistence mapping from entity beans, Hibernate or JDBC source.
- Visual Persistence diagram builder.
- Migration of your existing EJB projects (versions 1.x and 2.x) to EJB 3.0 with:
  - Converting EJB environment access.
  - Rebuilding EJB deployment descriptors.
  - Transforming EJB interfaces.
  - Turning Entity Beans into Container Managed Persistence.

Enabling EJB Support

You can enable EJB support when creating a new project or module. You can also add the EJB support for an existing module.

- Enabling EJB support when creating a project or module
- Enabling EJB support for an existing module

Enabling EJB support when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Java Enterprise . In the right-hand part of the page, specify the JDK to be used and select the Java EE version to be supported.

3. Under Additional Libraries and Frameworks, select the EJB: Enterprise Java Beans check box.

4. Select the EJB version to be supported from the Versions list.
   If you want an EJB deployment descriptor file `ejb-jar.xml` to be created, select the Create ejb-jar.xml check box.

   Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download and use an EJB library.
     To do that, under Libraries, select Download.

     Now, to view or modify the associated options, click Configure, and in the Downloading Options dialog that opens:
     - Specify the library name.
     - Select the library level (global, project, or module).
     - Under Files to download, select which of the files you want to download.
     - Under Copy downloaded files to, specify the path to the destination folder. If you want to change the default path, click and specify the folder location in the dialog that opens .
   - Use an EJB library IntelliJ IDEA is already aware of.
     To do that, click Use library and select the required library from the list.

     If necessary, configure the library settings (for example, change its name). This is done in the Edit Library dialog which you can open by clicking Configure.
   - Create a new library using the appropriate JAR files available on your computer.
     To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (For multiple selection, keep the `Ctrl` key pressed.)

     If necessary, configure the new library (for example, change its name or level). To do that, click Configure and specify the required settings in the Create Library dialog.
   - Postpone setting up the library until a later time. In this case, select Set up library later.

   Click Next.

5. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Enabling EJB support for an existing module

1. Open the Project tool window (e.g. View | Tool Windows | Project).

2. Right-click the module and select Add Framework Support.

3. In the left-hand pane of the Add Frameworks Support dialog, select the EJB: Enterprise Java Beans check box.
   Select the EJB version to be supported from the Versions list.

   If you want an EJB deployment descriptor file `ejb-jar.xml` to be created, select the Create ejb-jar.xml check box.

   Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download and use an EJB library.
     To do that, under Libraries, select Download.

     Now, to view or modify the associated options, click Configure, and in the Downloading Options dialog that opens:
     - Specify the library name.
     - Select the library level (global, project, or module).
     - Under Files to download, select which of the files you want to download.
     - Under Copy downloaded files to, specify the path to the destination folder. If you want to change the default path, click and specify the folder location in the dialog that opens .
   - Use an EJB library IntelliJ IDEA is already aware of.
     To do that, click Use library and select the required library from the list.

     If necessary, configure the library settings (for example, change its name). This is done in the Edit Library dialog which you can open by clicking Configure.
   - Create a new library using the appropriate JAR files available on your computer.
     To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (For multiple selection, keep the `Ctrl` key pressed.)

     If necessary, configure the new library (for example, change its name or level). To do that, click Configure and specify the required settings in the Create Library dialog.

– Postpone setting up the library until a later time. In this case, select Set up library later.

4. Click OK in the Add Frameworks Support dialog.

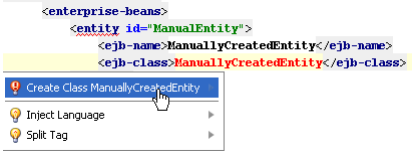
Creating EJB

IntelliJ IDEA provides the following ways to create beans:

- Manually, by means of direct editing the `ejb-jar.xml` file, using the extensive coding assistance provided by IntelliJ IDEA.
- Using the context menu commands.

To create an Enterprise Java bean using the editor

1. Open the deployment descriptor file `ejb-jar.xml` for editing.
2. Create the `<enterprise-beans>` section.
3. In this section, create tags for the desired bean types: `<session>`, `<entity>`, or `<message-driven >`
4. Specify the bean name and class by adding tags `<ejb-name>` and `<ejb-class>`. Note that you can specify the target package in this tag, for example, `samples.ejb.ManuallyCreatedEntity`, using code completion after each dot.
5. If the declared class does not yet exist, IntelliJ IDEA suggests a quick fix:



   Choose Create Class <class name> from the suggestion list. If the target package was not defined in the `<ejb-class>` tag, select the desired package in the Choose Destination Directory dialog box. The stub class is created in the specified location.
6. Proceed with defining the other bean components.

To create an Enterprise Java bean, follow these general steps

1. Open the EJB tool window.
2. Right-click the desired module, choose New on the context menu, and then select the bean type on the submenu.
   The New <bean type> Bean dialog box opens.

   Tip Alternatively, choose Jump to Source on the context menu of a module, to open the dedicated editor, in the General tab click New, or right-click the diagram background in the EJB Relationships tab, and select the desired bean type.
3. Specify the bean name, class and package.
4. Define the bean-specific settings:
   - For Entity beans (1.x and 2.x) and Session beans (1.x and 2.x), configure local and remote interfaces.
   - For Entity beans (1.x and 2.x), configure primary keys and CMP bean fields.
   - For Message beans, configure message listener.
   - For Session beans, configure service endpoint.

Configuring Primary Key

Primary keys are defined for CMP Entity beans and BMP Entity beans, versions 1.x and 2.x.

Same as the other bean properties, the primary keys can be defined by editing the source code, or using the tools provided by IntelliJ IDEA.

To configure a primary key

1. Create a new entity bean .
   Alternatively, open the desired bean for editing , and click the Change EJB Classes button.
2. In the Primary key class field specify the class that will be used to access the primary key of a datasource the bean is associated with.
3. From the CMP version list select EJB specification version for the bean. This setting is disabled for BMP entity beans.

Configuring Service Endpoint

Service endpoint is defined for Session beans, versions 1.x, 2.x and 3.0.

Same as the other bean properties, the service endpoint can be defined by editing the source code, or using the tools provided by IntelliJ IDEA.

To configure a service endpoint

1. Open a session bean for editing , and click Change EJB Classes.
2. In the Change EJB Classes dialog box, select the Service Endpoint check box and specify the to be used as an endpoint for calling services.

   Note For bean specification prior to 3.0 you also need to configure interfaces .

Creating and Editing Assembly Descriptors

Assembly Descriptors define how a bean is deployed and configured at the application server.

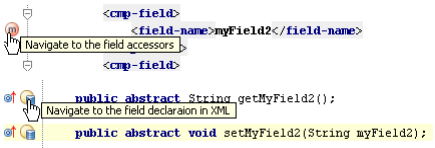
Same as the other bean properties, the application descriptors can be defined by editing the source code, or using the tools provided by IntelliJ IDEA.

To add an assembly descriptor to a bean

1. Open the desired bean for editing , and select the Assembly Descriptor tab.
2. Click + on the toolbar.
3. From the drop-down list, select the desired assembly descriptor type.
4. In the dialog box that opens, specify the assembly descriptor properties. Refer to the EJB Editor - Assembly Descriptor page for the detailed description of options.

Creating CMP Bean Fields

CMP fields are used to provide persistence of the entity beans. Being created, a CMP field appears in the deployment descriptor, and its accessor methods are added to the entity bean classes. So doing, IntelliJ IDEA provides gutter icons in the editor that help you jump from a CMP field declaration in the deployment descriptor to the accessor methods in the bean class, and vice versa:

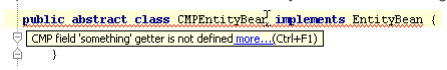


IntelliJ IDEA enables you to create CMP fields in several ways:

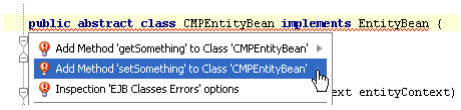
- By means of direct editing the source code . In this case, IntelliJ IDEA provides coding assistance, and the corresponding nodes are created in the EJB tool window.
- Using the context menu of an entity bean .
- Using the bean editor .

To create a CMP field by editing the source code

1. Open the deployment descriptor file `ejb-jar.xml` for editing.
2. In the `entity` section for the corresponding entity bean, type the tags for CMP fields, and specify their names.
3. Open the source code of the desired entity bean class for editing. The code inspection detects missing accessor methods:



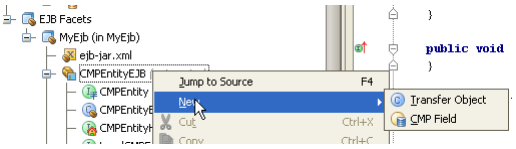
4. Press `Alt+Enter` to reveal the list of suggested quick fixes:



5. Create getter and setter methods in the bean implementation class.
6. If you want to make the new CMP field a primary key, add the <primkey-field> entry to the deployment descriptor.

To create a CMP field

1. In the EJB tool window, right-click the desired entity bean.
2. On the context menu, click New, and then choose CMP field on the submenu.



Tip Alternatively, open the desired bean for editing , and in the CMP Fields section click +.

3. In the Create CMP Field dialog box, specify the following parameters:
   - Field name, optional description and type. You can choose the desired type from the drop-down list, or click the ellipsis button and select type from the Choose CMP Field Class dialog box.
   - Whether the new CMP field will be a primary key. Select the Primary key check box, if you want to make this field a primary key.
   - Whether the accessor methods will be generated in the local and remote interfaces. If you select the corresponding check boxes, the getter and setter methods will be created in the local and remote interfaces, in addition to the bean class.
4. Click OK.

Creating Local and Remote Interfaces

Local and remote interfaces are used to provide access to the bean from the calling clients, and are defined for the following types of beans:

- Entity Beans, both CMP and BMP versions 1.x and 2.x
- Session Beans versions prior to 3.0

IntelliJ IDEA provides the following ways to create local and remote interfaces:

- Manually, by means of direct editing the `ejb-jar.xml` file, using the extensive coding assistance provided by IntelliJ IDEA.
- Using the context menu commands.

To define local and remote interfaces of an enterprise bean using the editor

1. Open the deployment descriptor file `ejb-jar.xml` for editing.
2. In the `<entity>` section, create tag `<local>`, or `<remote>` and type the name of the desired interface. Note that you can specify the target package in this tag, for example, `samples.ejb.ManuallyCreatedEntityLocalInterface`, using code completion after each dot.
3. If the declared class does not yet exist, IntelliJ IDEA suggests a quick fix. Choose Create Class <class name> from the suggestion list. If the target package was not defined in the `<ejb-class>` tag, select the desired package in the Choose Destination Directory dialog box. The stub class is created in the specified location.

To configure local and remote interfaces

1. Open the New Bean dialog box.
2. If you need to configure remote client view of a bean, select the Remote Interface check box.
   - In the Home field specify the name for the bean remote home interface.
   - In the Remote field specify the name for the bean remote interface.
3. If you need to configure local client view of a bean, select the Local Interface check box.
   - In the Home field specify the name for the bean local home interface.
   - In the Remote field specify the name for the bean local interface.

Creating Message Listeners

Message listener is defined for a Message bean, versions 1.x, 2.x or 3.0.

Same as the other bean properties, the message listener can be defined by editing the source code, or using the tools provided by IntelliJ IDEA.

To configure a message listener

1. Open the New Bean dialog box .
   Alternatively, open the message desired bean for editing , and click the Change EJB Classes button.

2. In the Message Listener field specify class that will be used to handle the Java Messaging Service messages.

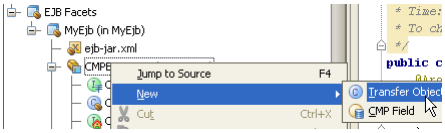
Creating Transfer Objects

Transfer Object is used as a proxy between an entity bean and its client. Entity beans use transfer objects to send data to the clients thus reducing the exchange traffic, because all required data are transferred at once, packed in a single object.

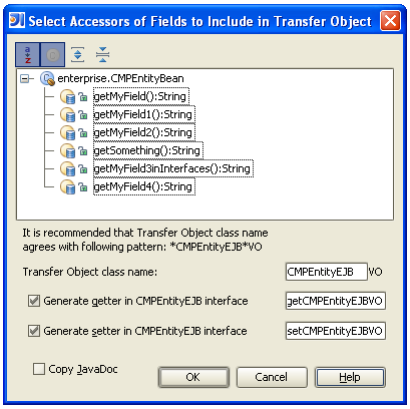
Though you can create a transfer object manually, as a class that implements the `Serializable` interface and provides accessor methods to certain fields of an entity bean, IntelliJ IDEA helps automatically generate stub source code.

To create a transfer object

1. In the EJB tool window, right-click the desired entity bean.
2. On the context menu, click New, and then choose Transfer Object on the submenu.



3. In the dialog box Select Accessors of Fields to Include in Transfer Object, select the methods to be included in the new transfer object.



Tip Use `Ctrl` and `Shift` keys in combination with the mouse click to select multiple adjacent or non-adjacent methods.

4. Specify the following parameters:
   - Name of the transfer object class. Note that IntelliJ IDEA suggests a pattern for generating the class name, so you can only change the initial part of it.
   - Whether the accessor methods will be generated in the bean interface.
5. Click OK.

Defining Bean Class and Package

You can define the bean class and package by means of editing the source code, or use the tools provided byIntelliJ IDEA.

To define bean class and package

1. Open the New Bean dialog box .
   Alternatively, open the desired bean for editing , and click the Change EJB Classes button.

2. In the <ejb-name > field, specify the bean name that will be complemented with a prefix and suffix configured in the Java EE Names tab of the Global Code Style dialog box.

3. In the Package field, specify the name of the package where the bean will reside. In the EJB class field, specify class name for this bean.

Editing Module with EJB Facet

Though you can directly edit the source code of the deployment descriptor of a module with EJB facet, it is also possible to use a dedicated editor , where you can manage the contained beans, their relationships and security permissions.

Note You can only edit the modules that contain `ejb-jar.xml` descriptor. For annotations-only modules, edit the annotations manually to adjust the module settings. If both annotations and `ejb-jar.xml` descriptor are present in a module, the descriptor settings override annotations.

To edit a module with EJB facet

1. Open the EJB tool window .
2. Right-click the desired module, and do one of the following:
   - Choose Jump to Source on the context menu.
   - Press F4 .
   - Double-click the EJB module node.

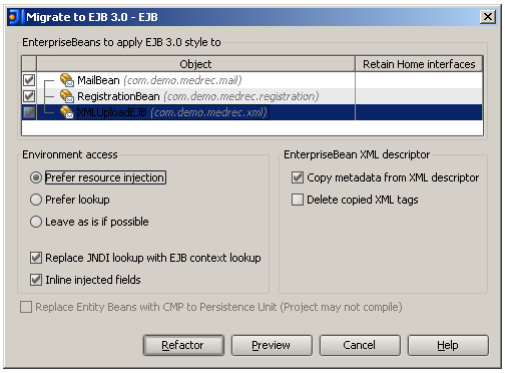
   Tip Same way you can also open for editing the `ejb-jar.xml` file. In this case, in addition to the General, Method Permissions, and Transaction Attributes tabs, the editor provides the Text tab, with the EJB-aware XML editor.
3. On the General tab of the editor, add new beans , edit existing beans , or delete them.
4. On the Method Permissions tab , configure security roles.
5. On the Transaction Attributes tab , specify transaction attributes for the bean methods.

Migrating to EJB 3.0

If you want to use the advantages of EJB 3.0 specification, you can migrate your existing beans of the various types from EJB 1.x or 2.0 to EJB 3.0. IntelliJ IDEA suggests the Apply EJB 3.0 Style refactoring that brings the structure of the beans into compliance with the requirements of EJB 3.0 specification.

To migrate a bean to EJB 3.0 specification

1. Open the EJB tool window .
2. Select the desired module, or bean, to be migrated, and choose Apply EJB 3.0 Style on the context menu.
3. In the Apply EJB 3.0 Style dialog box displaying the list of all items within the module, select the ones to be migrated:



If you want to convert home interfaces rather than remove them, select the Retain Home INterfaces check box, where applicable.

4. Configure the environment access and XML descriptor options. Refer to the Apply EJB 3.0 Style dialog for the detailed description of options.
5. Though you can click Refactor and perform migration straight away, it is recommended to click Preview first, and explore the tentative migration results in the Find Tool Window .
6. In the Find tool window, click Do Refactor.

Using EJB ER Diagram

EJB entity-relationship (ER) diagrams let you view entity beans, and also create and edit relationships between them.

To select elements in an EJB ER diagram

- To select an element, just click it on the diagram.
- To select multiple adjacent elements, keep the `Shift` key pressed, and click the desired elements, or just drag a *lasso* around the elements to be selected.
- To select multiple non-adjacent elements, keep `Ctrl+Shift` pressed and click the desired elements.

To manage the diagram layout

- Right-click the diagram background, and choose Layout command in the diagram context menu. Next, select the desired layout from the submenu.
- Use Drag-and-drop technique to lay out entities in the diagram manually.
- Restore the last selected layout by clicking on the diagram toolbar.

Erlang

IntelliJ IDEA provides Erlang support. IntelliJ IDEA recognizes `*.erl` files, and allows you to edit them providing full range of coding assistance. Erlang files are marked with icon.

In this section:

– Getting Started with Erlang

Getting Started with Erlang

Erlang is a great language that lets you build highly concurrent applications. This tutorial will teach you how to quickly get started with it.

In this section:

- Preliminary steps
  - Installing Erlang OTP
    - Windows
    - OS X
    - Linux
  - Verifying Erlang OTP installation
  - Installing Rebar
- Setting up IntelliJ IDEA
  - Configuring an Erlang SDK
  - Configuring Rebar
- Creating a new project
  - Creating an Erlang project
  - Creating a Rebar project
  - Importing a project into IntelliJ IDEA
- Running and debugging an application
- Running Eunit tests
- Running Rebar commands
- Additional
  - Learning Erlang
  - Learning IntelliJ IDEA
  - Providing Feedback

Preliminary steps

Installing Erlang OTP

The first thing for setting up an Erlang environment is installing Erlang OTP, a set of Erlang libraries essential for development.

Windows

If you are a Windows user, download the Erlang OTP package and run the installation wizard. Once the installation is over, add the installation path plus `\bin` to the PATH environment variable.

OS X

If you are an OS X user, to install Erlang OTP, type the following at the Terminal prompt (make sure you have Homebrew installed on your machine):

```
brew install erlang
```

Note that if you prefer MacPorts to Homebrew, your command line should be different:

```
port install erlang
                    +ssl
```

Linux

The installation process for Linux is similar to OS X, except that instead of “brew” or “port” you have to use “apt-get” (a Linux package management utility):

```
apt-get install
                    erlang
```

Note that you can always download the latest version of Erlang OTP package for any OS.

Verifying Erlang OTP installation

To verify that Erlang OTP is installed correctly, run the Erlang shell by typing `erl` in a Terminal prompt:

To learn more about the Erlang shell, you can read its user guide.

Installing Rebar

In addition to Erlang OTP, you’ll also need Rebar, a build tool that helps compile and test Erlang applications. The easiest way to install it on your machine is to download its sources and build it locally:

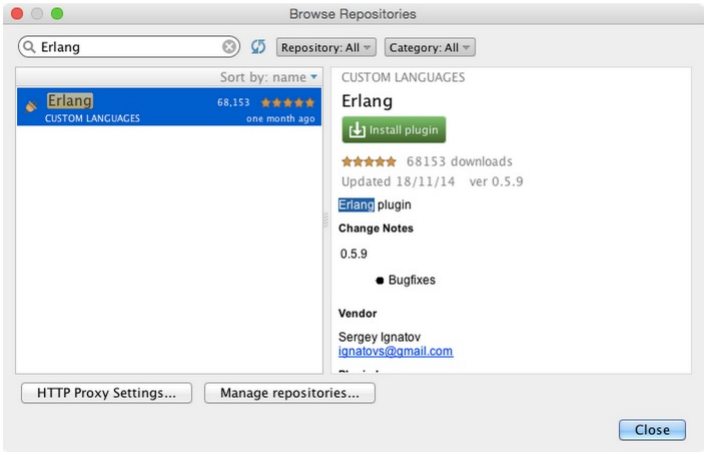
```
git clone git://github.com/rebar/rebar.git
                    $ cd rebar
                    $ ./bootstrap
                    Recompile: src/getopt
                    ...
                    Recompile: src/rebar_utils
                    ==> rebar (compile)
```

Congratulations! You now have a self-contained script called "rebar" in your current working directory. Place this script anywhere in your path and you can use rebar to build OTP-compliant apps.

Setting up IntelliJ IDEA

Now when Erlang OTP and Rebar are set up, it's time to download and install IntelliJ IDEA. Keep in mind, that for Erlang development you can use IntelliJ IDEA Community Edition (which is free and open-source).

Once the IDE is up and you see its Welcome screen , go to Configure | Plugins, then click Browse repositories, locate the Erlang plugin and install it:



After installing the plugin, restart IntelliJ IDEA.

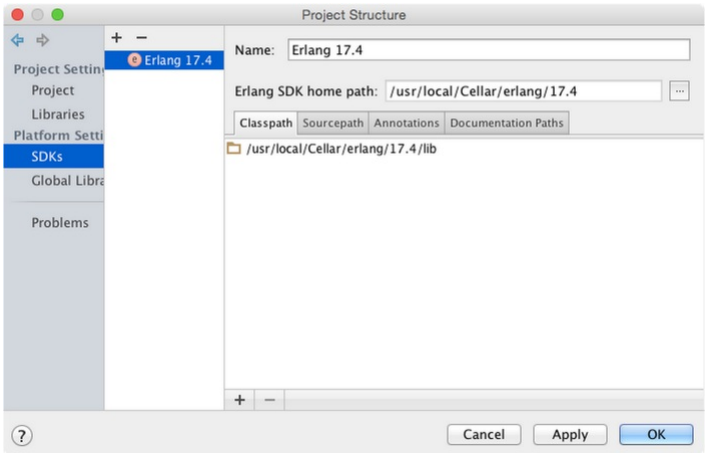
Configuring an Erlang SDK

One more thing you'll have to do to configure IntelliJ IDEA is to add an Erlang SDK.

To do that, change the structure of the default project. Open the default project structure in one of the two ways:

- On the Welcome screen , go to Configure | Project Defaults | Project Structure
- On the main menu, choose File | Other Settings | Default Project Structure

Then, add an Erlang SDK by specifying the path to the Erlang OTP installation directory.



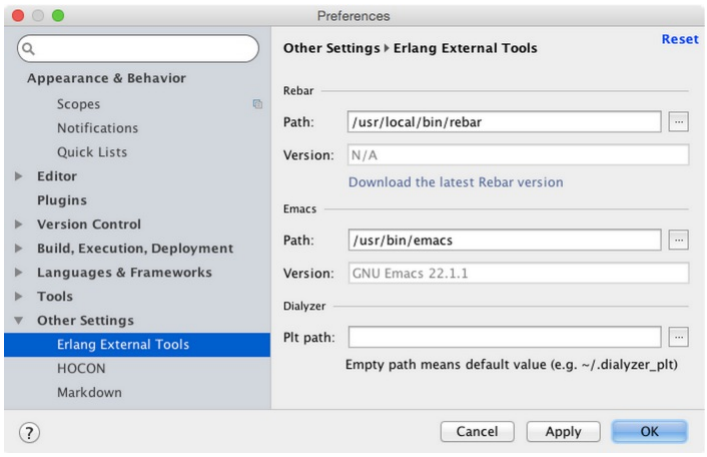
If you don't know where Erlang OTP was installed, check the following directories:

- Windows: `C:\Program Files\erl<version>`
- Linux: `/usr/lib/erlang/<version>`
- MacPorts, OS X: `/opt/local/lib/erlang/<version>`
- Homebrew, OS X: `/usr/local/Cellar/erlang/<version>`

Configuring Rebar

The final adjustment you have to do is to specify the path to Rebar, so that IntelliJ IDEA can run Rebar commands from the IDE.

You can do it via Configure | Preferences | Other Settings → Erlang External Tools:



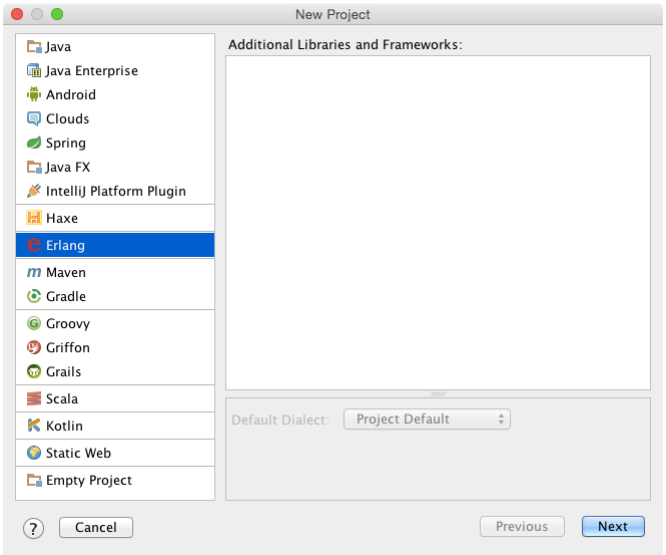
Creating a new project

Creating an Erlang project

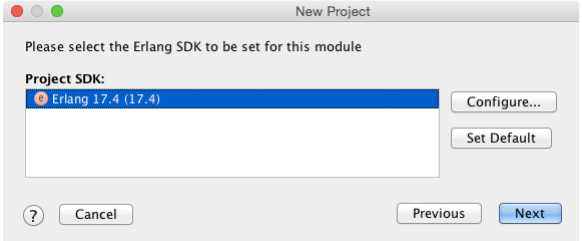
There are several ways to create a new Erlang project. The easiest one is to use the New Project Wizard from the Welcome screen.

Click Create New Project:

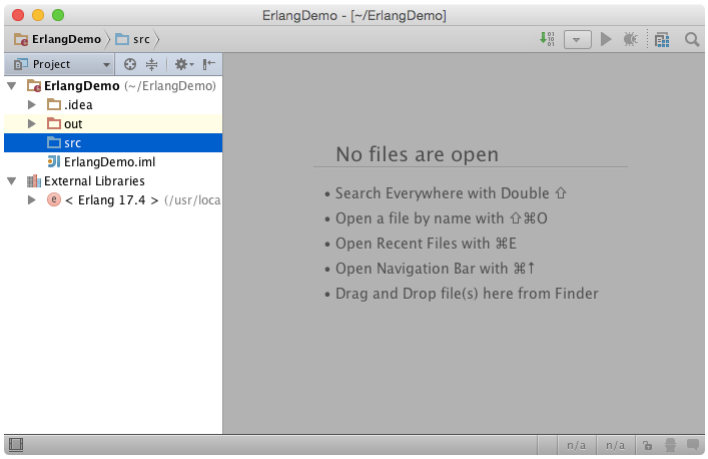
Then choose Erlang in the left pane, and click Next.



IntelliJ IDEA prompts you to choose an Erlang SDK (which you've already configured):



After that you'll be asked to specify the name of your project and its directory. The following image shows the resulting Erlang project with the name `ErlangDemo`:



Creating a Rebar project

Instead of a pure Erlang project, you might want to create a Rebar project. To do that, type the following code at the Terminal prompt:

```
rebar create-app appid=<project name>
```

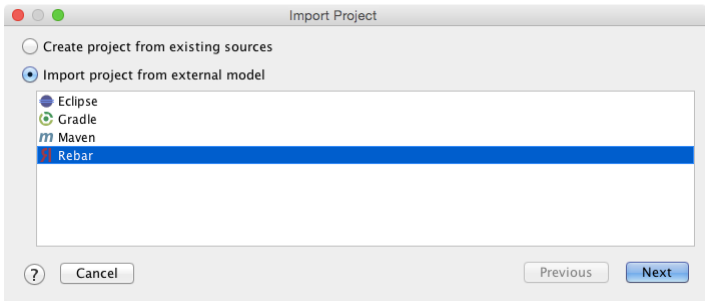
Once the project has been created, import it into IntelliJ IDEA to make it possible to open this project in the IDE.

Importing a project into IntelliJ IDEA

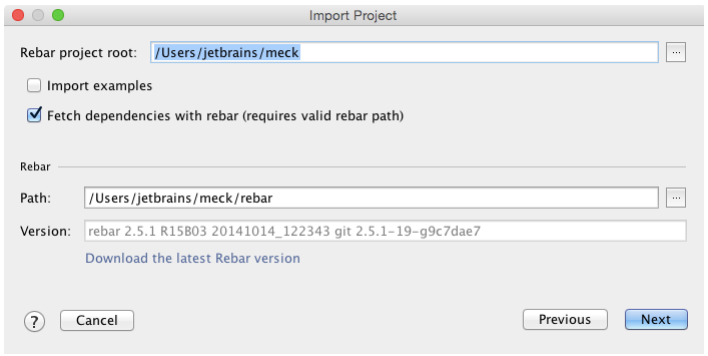
You can import a project into IntelliJ IDEA in several ways. Let's explore importing from the Welcome screen.

To import an existing project into IntelliJ IDEA, click Import on the Welcome Screen, and choose the project directory. IntelliJ IDEA offers you to either import the project from existing sources, or from an external model (a build file).

If your project uses Rebar, select the corresponding option when asked.

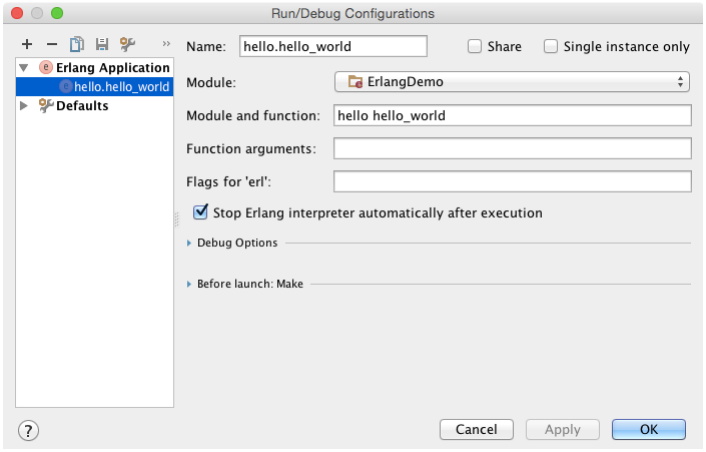


When importing a Rebar project, make sure to enable the option Fetch dependencies with rebar:



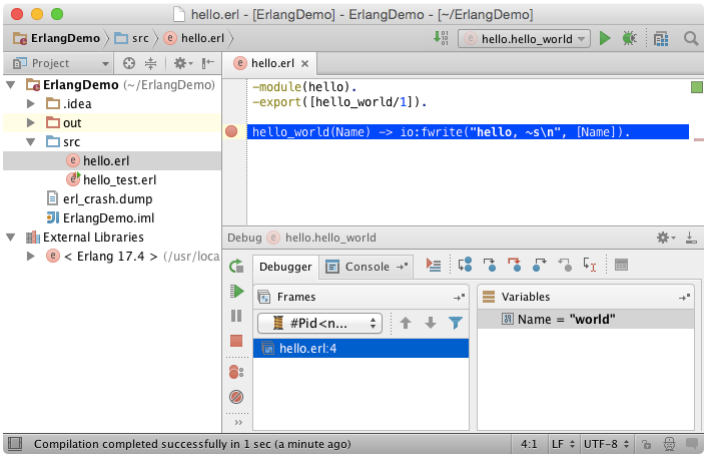
Running and debugging an application

To run an application, you have to create a run/debug configuration created against the stub Erlang Application. To do this, on the main menu choose Run | Edit Configurations, select the stub Erlang Application, specify the name (here it is `hello.hello_world`), and specify the application's module and function:



After that you'll be able to run your application via the main menu (Run | Run <run configuration name>, the toolbar ( ), or a even a shortcut (`Ctrl+Shift+F10`).

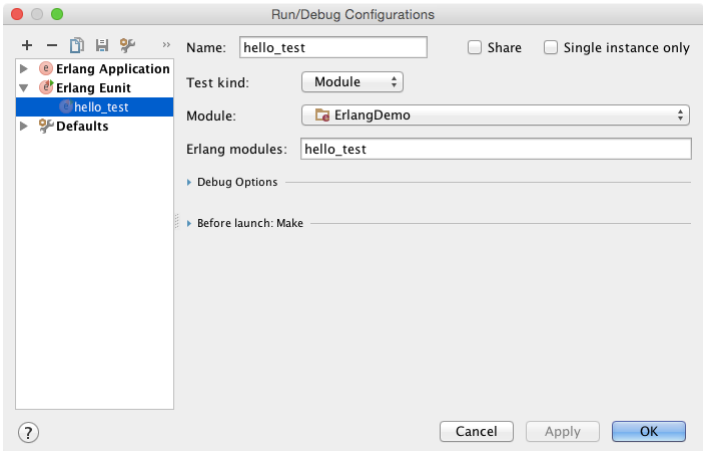
Once you have a run/debug configuration, you can also debug your application via the main menu (Run | Debug '<run configuration name>, the toolbar ( ), or a shortcut (`Shift+F9`):



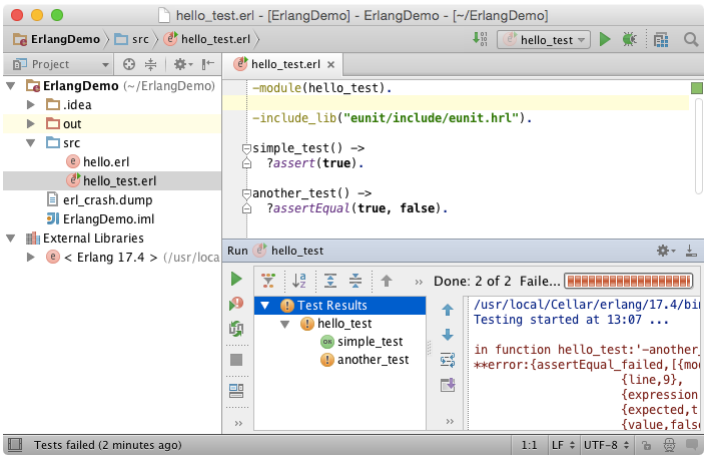
For more information, refer to the concept of a run/debug configuration and the procedural sections Running and Debugging .

Running Eunit tests

Running Eunit tests is similar to running an application, but needs a different run/debug configuration, created against the stub Erlang Eunit:

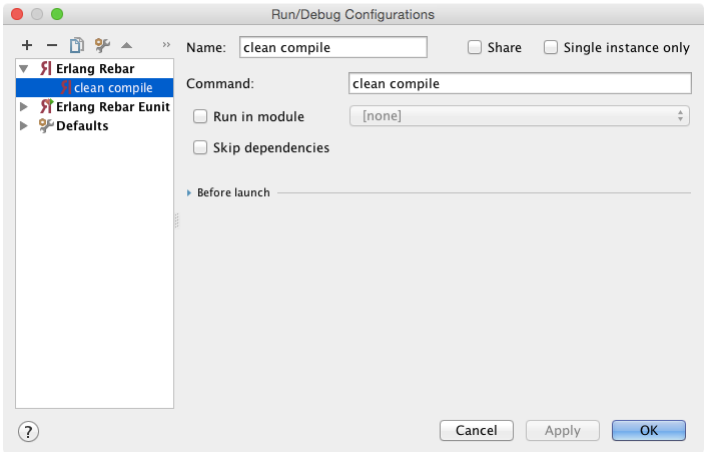


IntelliJ IDEA provides a handy Test Runner with support for Eunit . It shows test results, lets you rerun tests of you choice, jump to failed tests, etc.:



Running Rebar commands

Running Rebar commands is also possible right from the IDE – with the help of the Erlang Rebar run/debug configuration:



Note that if your Rebar commands run tests, you can use a Erlang Rebar Eunit run/debug configuration to see test results in a Test Runner.

Additional

Learning Erlang

To learn Erlang, we recommend you to start by reading the official Erlang user guide , and of course the Learn You Some Erlang for Great Good tutorial by Fred Hebert .

Learning IntelliJ IDEA

IntelliJ IDEA is a Java IDE in the first place, however it's also a platform and IDE for other languages, such as Erlang, Python, Ruby, PHP, and many other. To learn more about IntelliJ IDEA, it's worth checking out the IntelliJ IDEA Quick Start Guide and watch the Video Tutorials .

If you have a question, you can always ask it on StackOverflow (probably it's already answered).

Providing Feedback

In case you'd like to share your feedback about IntelliJ IDEA or its support for Erlang, feel free to submit an issue in Erlang plugin GitHub repository , or to the IntelliJ IDEA issue tracker .

Refer to the section Reporting Issues and Sharing Your Feedback .

Grails

IntelliJ IDEA implements Grails technology and allows creating Grails application with the specific folder structure and all the necessary artifacts.

IntelliJ IDEA supports all features of Grails 3.0.0 and later versions.

In this section:

- Grails
  - Prerequisites
  - Grails features in IntelliJ IDEA
  - Grails changes in the IntelliJ IDEA UI
- Getting Started with Grails 3
- Getting Started with Grails 1/2
- Creating a Grails Application Module
- Creating Grails Application from Existing Code
- Grails Procedures

Prerequisites

Make sure that the desired SDK is downloaded and installed on your computer, and the libraries are properly configured.

Also, make sure that the Grails plugin is enabled in IntelliJ IDEA. The plugin is bundled with IntelliJ IDEA and is activated by default. If the plugin is not enabled, enable the plugin .

Grails Features in IntelliJ IDEA

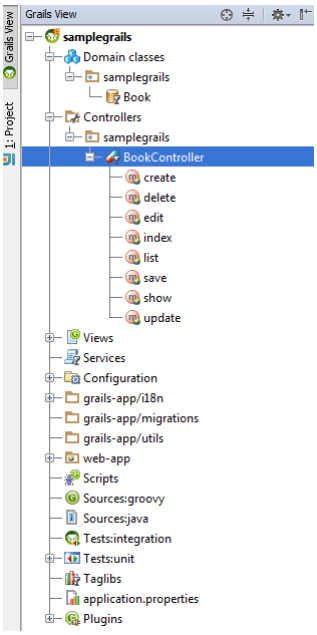
Grails support in IntelliJ IDEA includes the following features:

- Automated way of creating Grails Applications that provides generation of the specific structure and artifacts.
- Automatic enabling Groovy support in the Grails Application modules, allowing creation of the Groovy classes, interfaces and scripts.
- Execution of the targets .
- Possibility to generate scaffolding .
- Possibility to generate tests .
- Code completion .
- Querying with dynamic finders .
- Dedicated run/debug configuration .
- Grails 3.1.0 supports the GORM version 5.0.
- Grails Web Layer .
- Grails Resources plugin .
- Grails Spock .
- Grails Standalone GORM

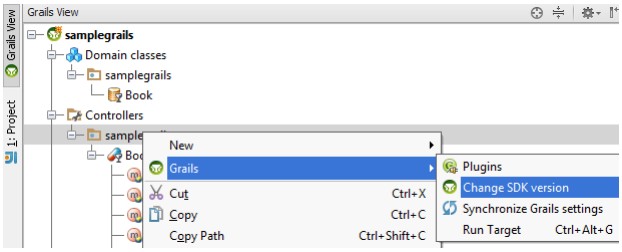
Grails Changes in the IntelliJ IDEA UI

Once IntelliJ IDEA recognizes the project or module as a Grails application, it introduces the following changes to the UI:

- Grails View tool window is added, and the tool window button appears in the left tool windows bar:

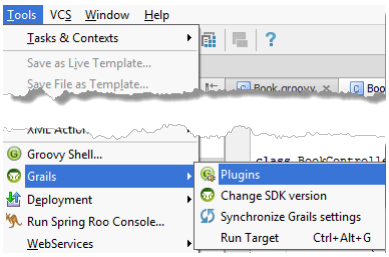


- Grails node is added to the context menu of the respective module in the Grails view:



– Note Please note that the Plugins option is available only for the earlier Grails versions up to the version 2.3.0.

Grails node is added to the Tools menu:



– Access to Groovy console in Grails applications (Tools | Groovy Shell).

Getting Started with Grails 3

IntelliJ IDEA supports Grails version 3.0 and later.

- Before you start
- Creating Grails 3 Project
- Exploring Grails Application
- Running Grails 3 Application
- Debugging Grails 3 Application
- Grails 3 Coding Assistance
- Grails 3 Gradle Support

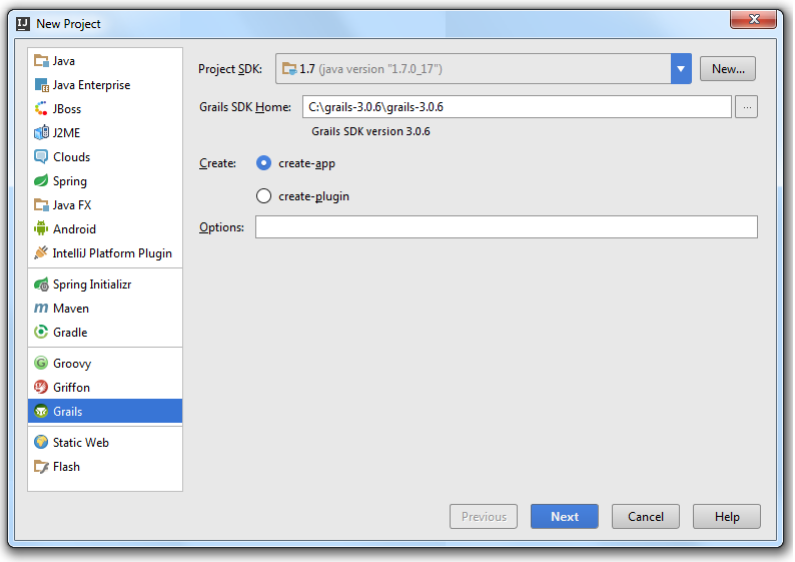
Before you start

Before you start creating your project with Grails framework, make sure that you have the latest Grails SDK version downloaded on your machine. You can download the latest SDK version from the Grails page. Also, make sure that you are working with IntelliJ IDEA ultimate edition, version 15 or higher if you want to work with the Grails 3.0 version. See the latest available version.

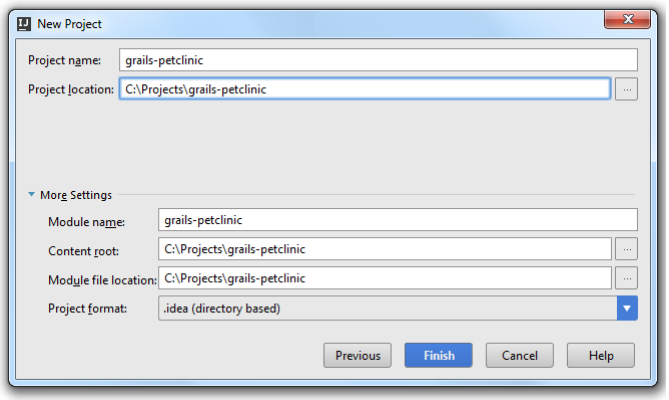
Creating Grails 3 Project

1. Open Project Wizard, in the left-hand pane select Grails.
2. In the right-hand part of the page, specify the following information:
   - Project JDK that you are going to use for your project.
   - Grails SDK Home - your local Grails 3 installation which is represented by a library.
   - Create create-app or create-plugin- select one of these options depending on what you want to create.
   - Options - use this field for additional options. For example, you can specify a profile such as `--profile=web` for Grails 3.0.

   Click Next.



3. On the next page of the wizard, specify the project's information and click Finish.

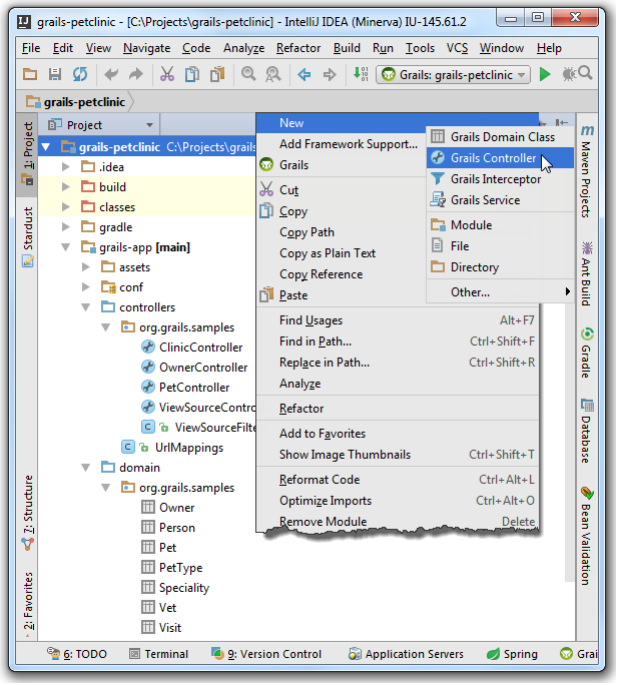


IntelliJ IDEA creates the Grails application.

Note For the Grails 3.0. version, IntelliJ IDEA automatically links the created project as a Gradle project with all Gradle functionality.

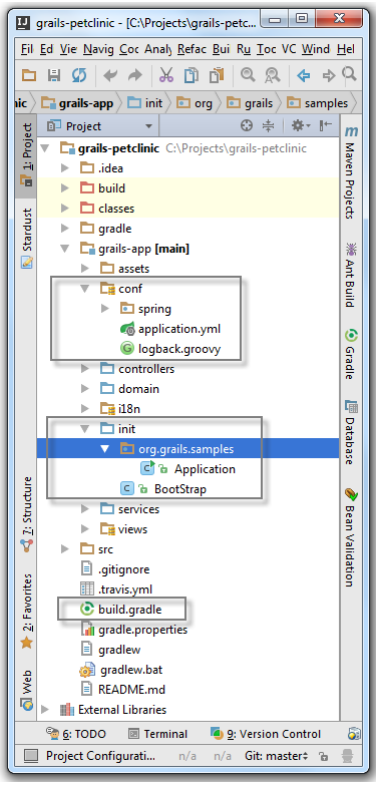
Exploring Grails Application

IntelliJ IDEA enables you to explore your Grails application. Note that the Grails View tool window is not supported for Grails 3. However, Grails actions are available in the Project View tool window. All artifact icons are changed to Grails icons. For Grails 1 and 2 versions the Grails View tool window is available.



Other notable differences from the previous Grails versions are as follows:

- `build.gradle` - the Grails version 3.0 uses Gradle for building. When you import Grails 3 project you can import it through the Gradle model.
- config directory - the content of the config directory enables you to use either a YAML file or Groovy file for your configuration. The logging configuration is also available.
- init directory - this directory contains main application file the lets you run your application with default settings.



For more information on the Grails version 3 changes, please see the Grails page.

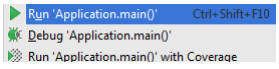
Running Grails 3 Application

When the Grails application is created it is ready to run.

Click the ▶ icon on the top right corner of your workspace.

If you want to run the application from the editor, perform the following steps:

1. In the Project tool window, click the init folder.
2. From the drop-down list, select Application.groovy to open the file in the editor.
3. In the editor, in the left gutter, click ▶ icon and in the window that opens click Run 'Application main()'.



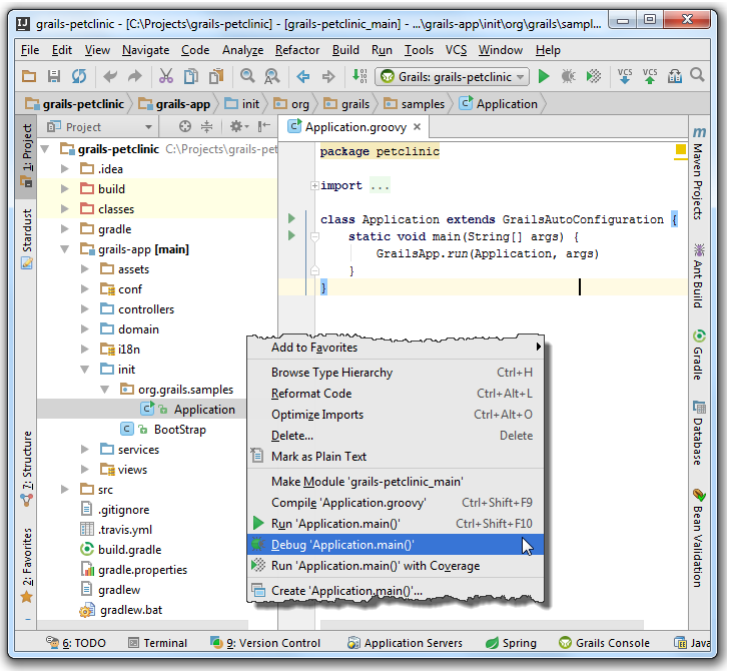
Your application starts in your default browser, with the following URL in the address bar:

```
http://localhost:8080/
```

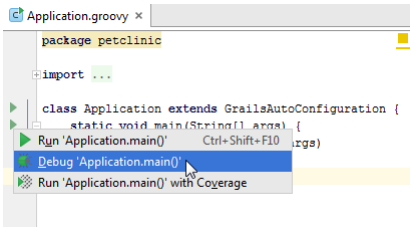
Debugging Grails 3 Application

IntelliJ IDEA lets you debug your Grails 3 application using Application.groovy.

1. In the Project tool window, open init directory and right-click the Application.groovy
2. From the drop-down list select Debug Grails:'name'

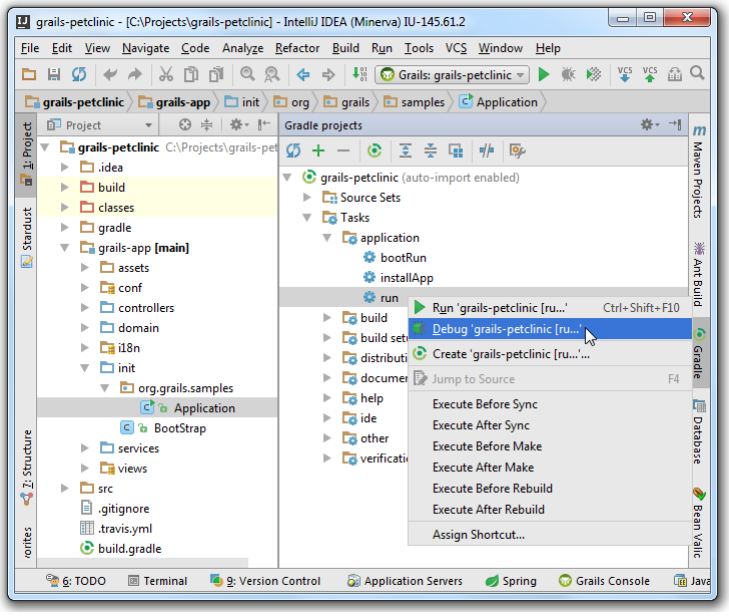


You can also use the editor to start the debugging process.



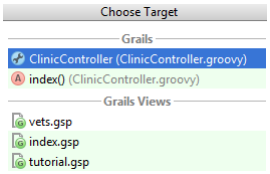
IntelliJ IDEA lets you also debug your Grails 3 application using Gradle tasks.

1. Open Gradle tool window .
2. From the list of tasks, click application and in the list that opens, right-click run.
3. From the drop-down list that opens, select Debug Grails:'name'

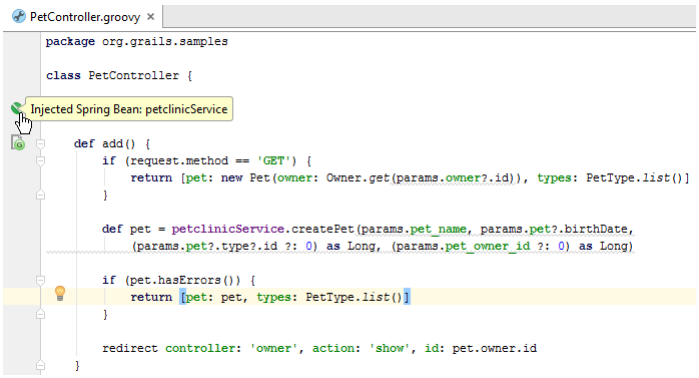


Grails 3 Coding Assistance

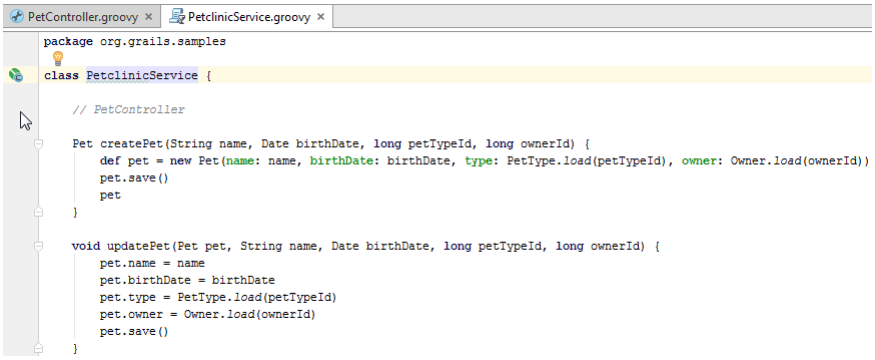
- IntelliJ IDEA provides navigation between all related files such as domain classes, views, services and controllers. Press `Ctrl+Alt+Home` in the editor and choose your target.



– IntelliJ IDEA provides navigation between the injected dependencies.

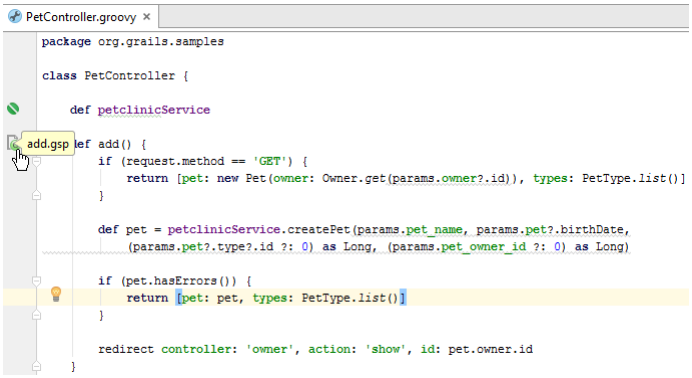



```groovy
package org.grails.samples

class PetController {

    def add() {
        if (request.method == 'GET') {
            return [pet: new Pet(owner: Owner.get(params.owner?.id)), types: PetType.list()]
        }

        def pet = petclinicService.createPet(params.pet_name, params.pet?.birthDate,
            (params.pet?.type?.id ?: 0) as Long, (params.pet_owner_id ?: 0) as Long)

        if (pet.hasErrors()) {
            return [pet: pet, types: PetType.list()]
        }

        redirect controller: 'owner', action: 'show', id: pet.owner.id
    }
```

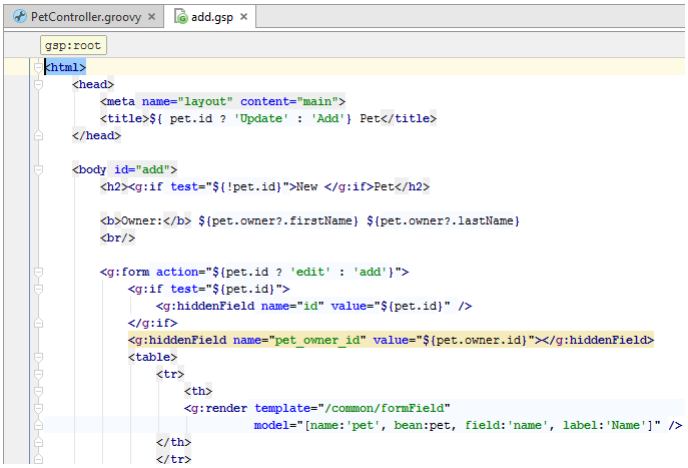



```groovy
package org.grails.samples

class PetclinicService {

    // PetController

    Pet createPet(String name, Date birthDate, long petTypeId, long ownerId) {
        def pet = new Pet(name: name, birthDate: birthDate, type: PetType.load(petTypeId), owner: Owner.load(ownerId))
        pet.save()
        pet
    }

    void updatePet(Pet pet, String name, Date birthDate, long petTypeId, long ownerId) {
        pet.name = name
        pet.birthDate = birthDate
        pet.type = PetType.load(petTypeId)
        pet.owner = Owner.load(ownerId)
        pet.save()
    }
```

– IntelliJ IDEA also provides navigation between the methods of the controller and `.gsp` files.

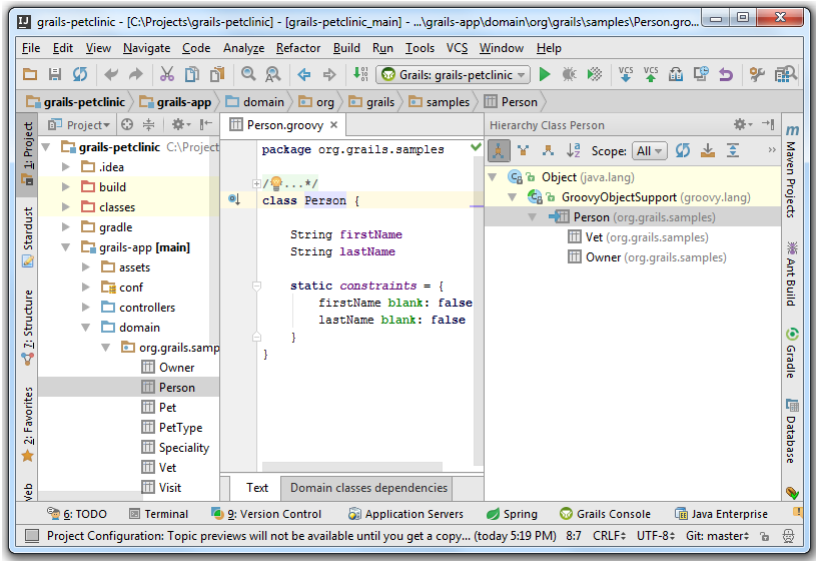



```groovy
package org.grails.samples

class PetController {

    def petclinicService

    def add() {
        if (request.method == 'GET') {
            return [pet: new Pet(owner: Owner.get(params.owner?.id)), types: PetType.list()]
        }

        def pet = petclinicService.createPet(params.pet_name, params.pet?.birthDate,
            (params.pet?.type?.id ?: 0) as Long, (params.pet_owner_id ?: 0) as Long)

        if (pet.hasErrors()) {
            return [pet: pet, types: PetType.list()]
        }

        redirect controller: 'owner', action: 'show', id: pet.owner.id
    }
```

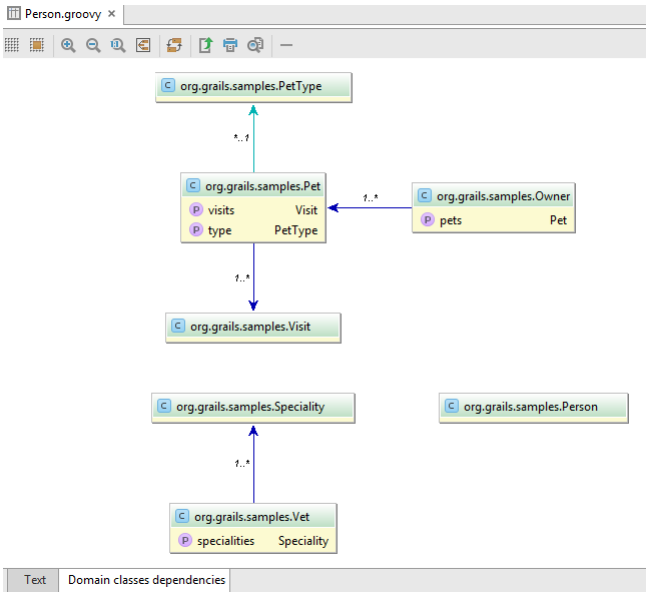



```
<html>
    <head>
        <meta name="layout" content="main">
        <title>${ pet.id ? 'Update' : 'Add'} Pet</title>
    </head>

    <body id="add">
        <h2><g:if test="${!pet.id}">New </g:if>Pet</h2>

        <b>Owner:</b> ${pet.owner?.firstName} ${pet.owner?.lastName}
        <br/>

        <g:form action="${pet.id ? 'edit' : 'add'}">
            <g:if test="${pet.id}">
                <g:hiddenField name="id" value="${pet.id}" />
            </g:if>
            <g:hiddenField name="pet_owner_id" value="${pet.owner.id}"></g:hiddenField>
            <table>
                <tr>
                    <th>
                    <g:render template="/common/formField"
                              model="[name:'pet', bean:pet, field:'name', label:'Name']" />
                    </th>
                    </tr>
```

– IntelliJ IDEA lets you open and check the hierarchy of the domain classes. Press `Ctrl+H` in the editor to open Hierarchy tool window.



– IntelliJ IDEA also lets you check the domain classes dependencies.



Grails 3 Gradle Support

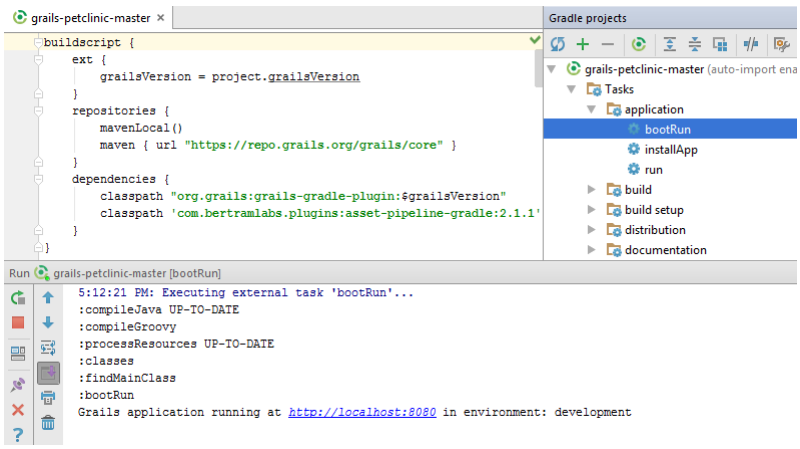
Grails 3 uses Gradle build system. You can use Gradle for the following actions:

– run Gradle tasks

– rely on the coding assistance when you edit build files

– import Grails 3 project from a Gradle model

– automatically update project dependencies



Getting Started with Grails 1/2

IntelliJ IDEA tightly integrates with Grails , and makes it possible to work with Grails applications from within the IDE, sparing you from the need to use a command line. Grails support in IntelliJ IDEA lets you do the following:

- Before you start
- Creating Grails Project
- Exploring Grails Application
- Creating Elements in your Grails Project
- Running the Application

Before you start

Before you start creating your Grails project, make sure that you have Grails SDK downloaded on your machine. You can download the latest SDK version from the Grails page . Also, make sure that you are working with IntelliJ IDEA ultimate edition, version 9 or higher. See the latest available version .

Warning! If you plan on using the Grails 3.0 version, please see the Getting Started with Grails 3 .

Creating Grails Project

Do one of the following:

- If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
  As a result, the New Project wizard opens.
- If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
  As a result, the New Module wizard opens.

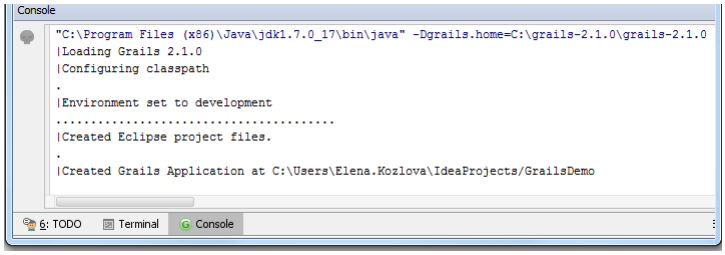
On the first page of the wizard, in the left-hand pane, select Grails. In the right-hand part of the page, specify the following setting:

- Project SDK that you are going to use.
- Grails SDK Home - select a local Grails installation that is represented by a library .
- Create create-app or create-plugin - select one of these options depending on what you want to create.
- Options - specify additional options.

Click Next.

Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
Click Finish.

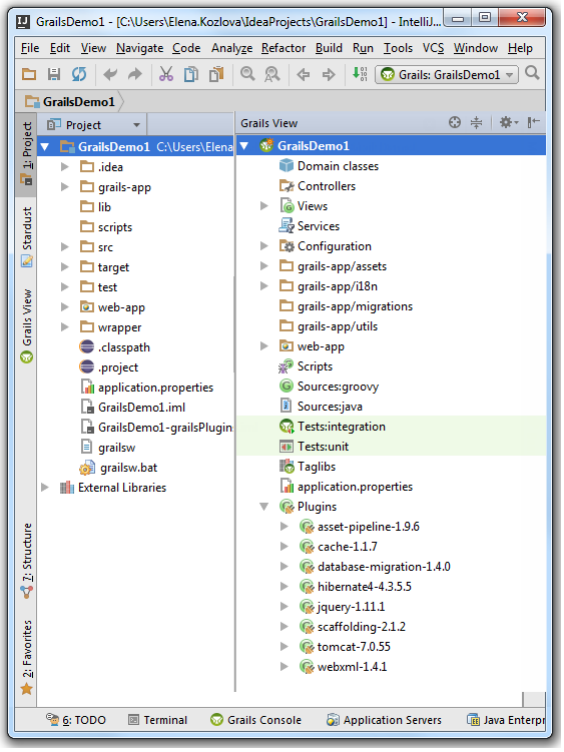
Since we chose to create an application, IntelliJ IDEA executes the create-app target, which generates the directory structure of a Grails application. All output information is displayed in the Console:



Exploring Grails Application

IntelliJ IDEA enables you to explore your Grails application from two different viewpoints:

- The Project tool window shows the typical Grails directory structure.
- The Grails tool window shows the logical set of Grails application elements (Domain classes, Controllers, Views, etc.)



Creating Elements in your Grails Project

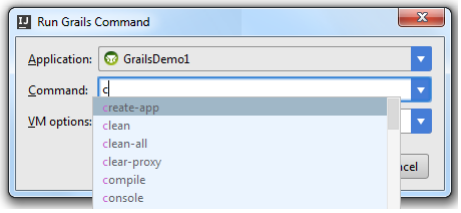
To illustrate IntelliJ IDEA abilities, let's start developing a very basic library management system.

1. Create a domain class for the library system. This class will represent a book within a library.

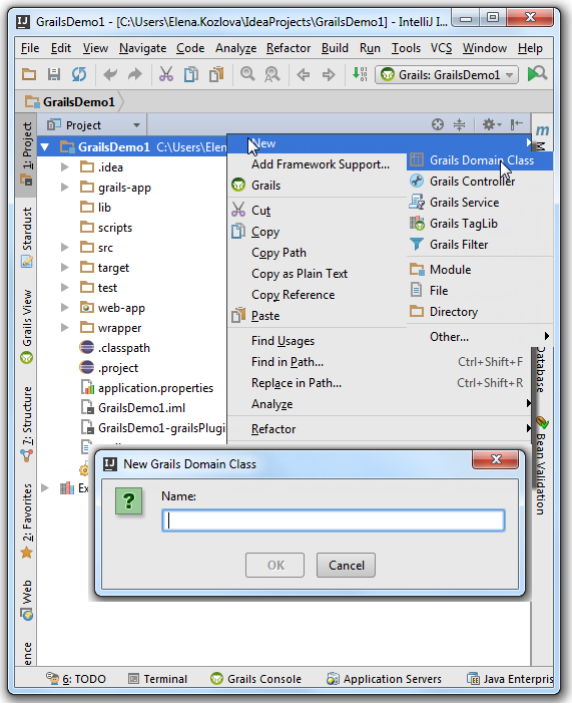
There are two possible ways of doing that in IntelliJ IDEA:

– Execute a Grails target. Press Ctrl+ Alt+ G, and enter Grails target name.

Note that code completion Ctrl+Space is available in the Run Grails target dialog box:



– Right-click the Grails tool window background, and choose New | Grails Domain class on the context menu:



As a result, two stub classes are created in the project:

– Domain class Book.groovy

– Test class BookTests.groovy

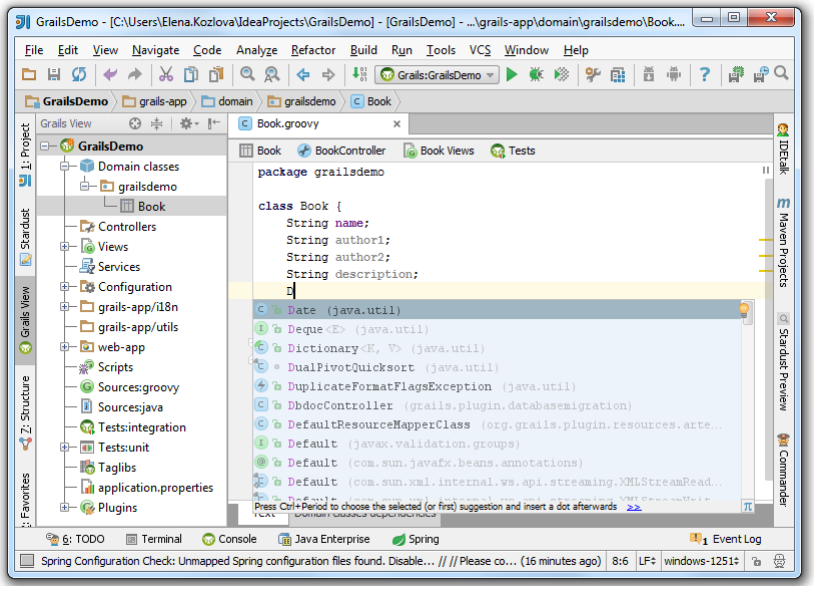
IntelliJ IDEA diligently shows all output messages in the Console.

For the purposes of our tutorial, we will work with the domain class Book.groovy.

Now it is just a stub, and we'll add the following fields:

- Book title
- Author name (may be two author names?)
- Description
- Publisher
- Date published
- Copyright
- ISBN
- Reader name
- Date taken

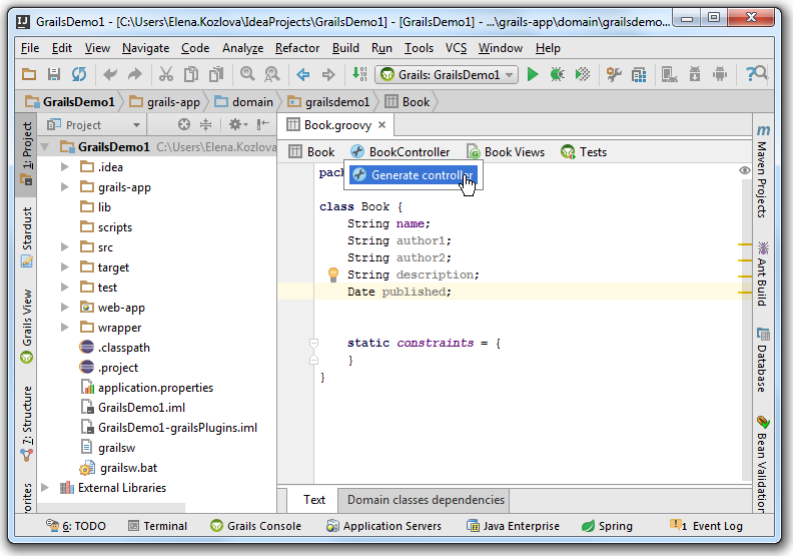
2. Open Book.groovy for editing (F4), and type these fields in your code, using the code completion Ctrl+Space:



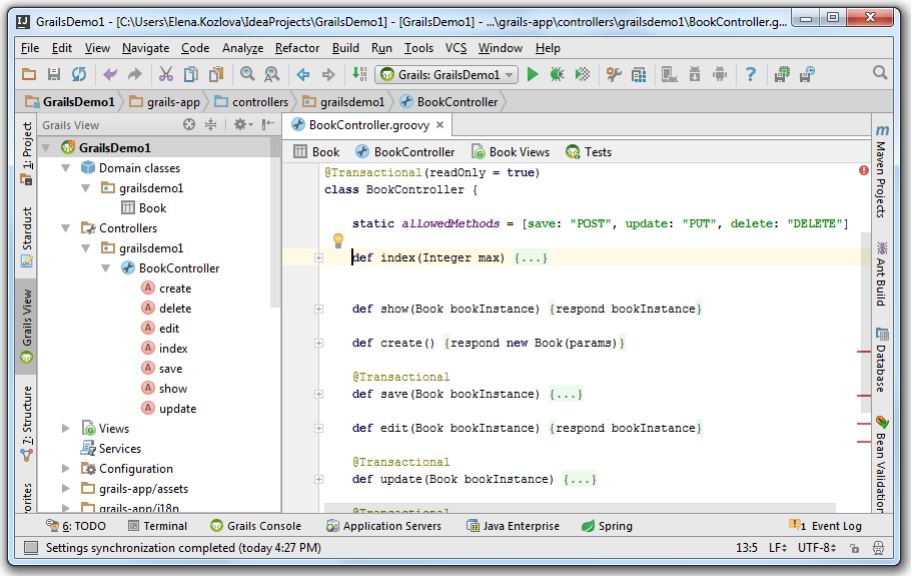
3. Provide a controller and views.

You can do it in two ways:

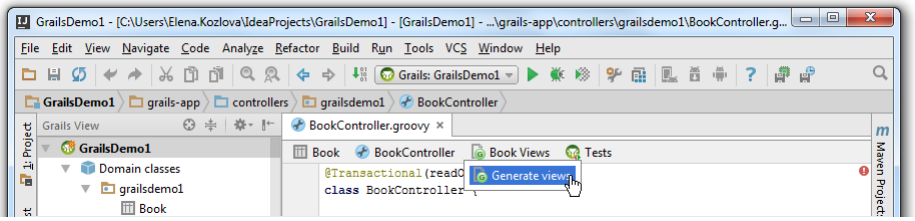
- run the Grails target generate-all Book
- use Scaffolding - the handy tool that you can find at the top of the domain class editor:



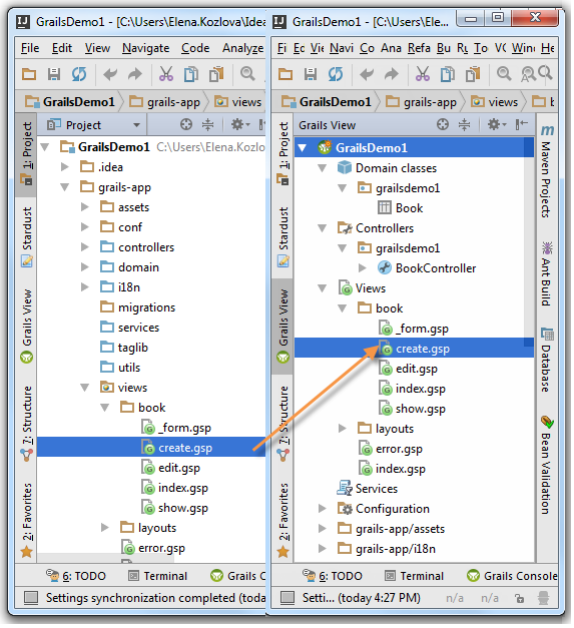
IntelliJ IDEA works hard (you can see that in the console), and produces the BookController.groovy class:



Next, create views the same way:



For each method of the controller, IntelliJ IDEA generates a file with the .gsp extension (create.gsp, edit.gsp, list.gsp, show.gsp).



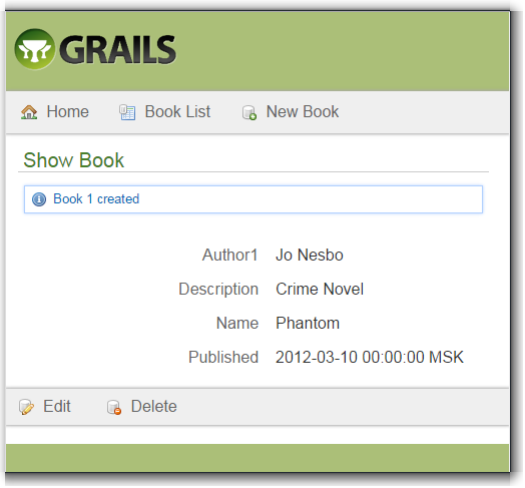
Running the Application

There are more things you might want to do to make your application useful, but let's try to run it straight away with the default settings. To do that, press Shift+F10 and after a turmoil of messages in the Console, your application starts in your default browser, with the following URL in the address bar: `http://localhost:8080/GrailsDemo`.

On that page, you will see something like this:

- grailsdemo BookController

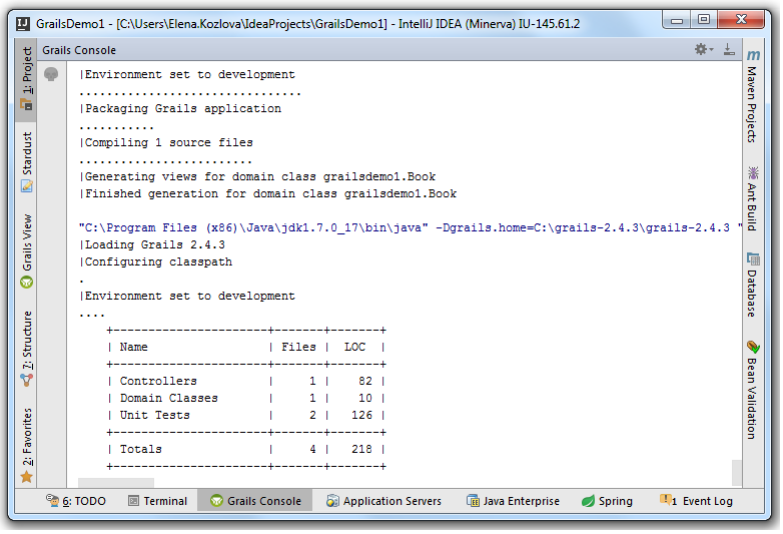
Click the controller link to open the list of books, which is empty by default. Now, you can try to fill out the entries of your library management system, for example, click the New Book button to add a book:



As you see, our basic library management system is ready. If you want to extend its functionality or you are not very happy with the code generated by Grails, you can modify the files in the IntelliJ IDEA editor to fit your particular needs, and rerun the application.

Finally, if you want to evaluate your effort for creating and running your Grails application under IntelliJ IDEA, view the number of files and lines of source code.

Press Ctrl+ Alt+ G, type `stats` in the pop-up window, and see the results in the Console:



Creating a Grails Application Module

You can create a new project with a Grails module or add a new Grails module to an existing project.

To create a Grails Application module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Grails. In the right-hand part of the page, specify the following setting:
   - Project SDK that you are going to use.
   - Grails SDK Home - select a local Grails installation that is represented by a library .
   - Create create-app or create-plugin - select one of these options depending on what you want to create.
   - Options - specify additional options.

   Click Next.

3. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Creating Grails Application from Existing Code

If you already have a Grails or Griffon project, you can create a IntelliJ IDEA Grails or Griffon Application around its source code. IntelliJ IDEA recognizes structure, marks source roots, and automatically enables Groovy-based frameworks.

To create application around a Grails or Griffon project

- Create a new project around existing code , pointing to the directory that contains the source Grails project. On the last page of the wizard that displays recognized technologies, select the ones you want to be enabled in the new application.

Grails Procedures

This section covers Grails- and Griffon-specific procedures:

- Creating Grails Application Elements
- Scaffolding
- Creating and Editing Relationships Between Domain Classes
- Creating Grails Views and Actions
- Navigating Between Actions and Views
- Running Grails Targets
- Running and Debugging Grails Applications
- Testing Grails Applications
- Working with Grails Plugins
- Dynamic Finders

Tip Note that Grails support is available in the Ultimate edition only.

Creating Grails Application Elements

IntelliJ IDEA enables easy creation of the Grails or Griffon Application elements in the modules of the corresponding type (domain classes, controllers, scripts etc.) You can create new elements using the application-specific Grails or Griffon tool windows, or the Project tool window.

Execution of the target that corresponds to the selected element type, is displayed in the console.

To add a new Grails or Griffon element

1. Open Grails or Griffon tool window, depending on the desired application type, and right-click the package where a new application element should be added.
2. On the context menu of the destination package, choose New, or press `Alt+Insert`, and choose the element type.
3. Type the name of the new element. So doing, you may not care about capitalization: when a new element is generated, its name is capitalized automatically. Click OK. A Groovy class of the selected type is created in the location stipulated by the application structure.
4. Add the necessary source code in the editor, using the Groovy-aware coding assistance, refactorings and intention actions.

IntelliJ IDEA provides the possibility to create Grails filters in one click. Just press `Alt+Insert`, and choose Grails Filter on the pop-up menu. After executing the `CreateFilters.groovy` target, the stub class with the filter definition is created under `grails-app/conf` directory of your Grails application.

Note that IntelliJ IDEA automatically provides the proper name of your filter class (`*Filters.groovy`), so you only need to specify the initial characters of the filter class name.

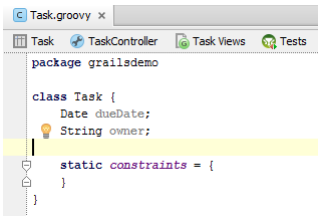
Scaffolding

IntelliJ IDEA implements the powerful Grails facility to generate controllers, views, and tests from the Grails elements. For this purpose, the editor for the Grails elements provides a toolbar, which enables you to run the Grails scaffolding generation targets.

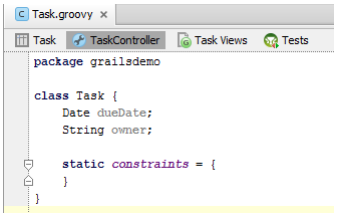
When you run a target, the names of the scaffolding elements are generated on the base of the Grails elements' names, and the scaffolding files are placed to the corresponding directories of the module structure.

To generate scaffolding for a Grails element

1. Open the desired Grails element in the editor:



2. Click the desired scaffolding button, and choose Generate:



Creating and Editing Relationships Between Domain Classes

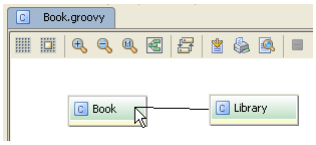
Use the Domain classes dependencies diagram for creating and editing relationships between the domain classes.

The diagram is completely synchronized with the source code: the changes in the source code are immediately reflected in the diagram, and vice versa, adding or deleting a link in diagram introduces relevant changes to the source code.

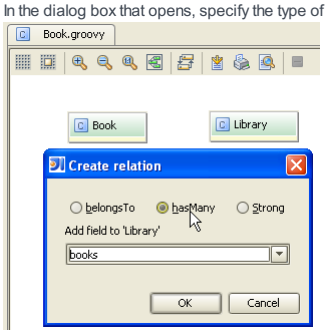
To create relationship between domain classes

1. Select the desired domain class, and click the Domain classes dependencies tab that is located on the bottom of the editor.

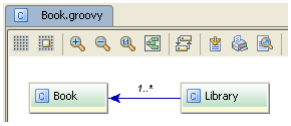
2. Click the source domain class and draw a link to the target domain class:



3. In the dialog box that opens, specify the type of relationship, and the name of the field to be created:



The resulting relationship is displayed in diagram:



Creating Grails Views and Actions

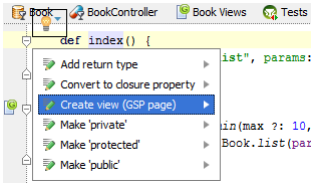
IntelliJ IDEA provides intention actions to create views and actions "from usage".

To create an action from a view

1. In a Grails `*.gsp` page, place the caret at a view that refers to a non-existent action.
2. Press `Alt+Enter`, and choose Create Action from the suggestion list. The action is created in the controller, with the caret at the insertion point.
3. Type the meaningful action code.

To create a Grails view from an action

1. In a Grails controller, place the caret at an action that refers to a non-existent view.
2. Press `Alt+Enter`, and choose Create View (GSP page) from the suggestion list.



A `.gsp` file with the corresponding name is created under the `views` directory of the Grails application.

Tip Configure code style of `.gsp` pages here .

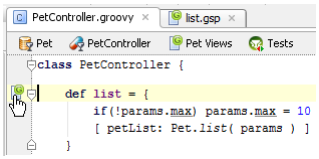
Navigating Between Actions and Views

IntelliJ IDEA suggests convenient means of navigation within Grails applications. The following cases are possible:

- Between a controller action and a view with the same name .
- From `redirect` and `render` methods to the corresponding controllers and actions .

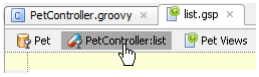
To navigate between an action and the corresponding view

- In a controller, with the caret within an action: Click the icon in the left gutter:



The corresponding `<name>.gsp` file opens in the editor.

- In a view file `<name>.gsp` : click the Controller button on the scaffolding toolbar :



The corresponding controller opens in the editor, with the caret resting before the action name.

To navigate from Grails render or redirect methods, do one of the following

- With the caret at the template name, press `Ctrl+B`
- Keeping `Ctrl` pressed, click the template name:

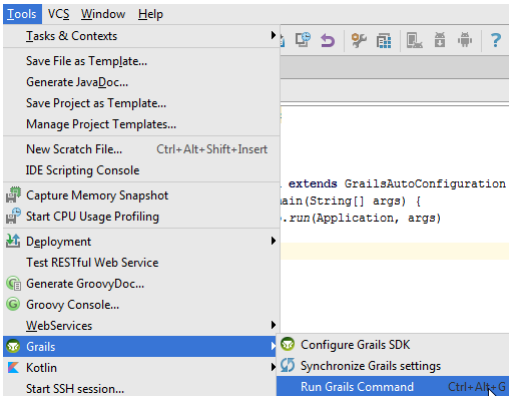
Running Grails Targets

IntelliJ IDEA enables launching targets in the Grails or Griffon applications. When Grails or Griffon application is selected, the respective Run target command appears in the Grails or Griffon nodes of the main Tools menu.

To run a Grails or Griffon target

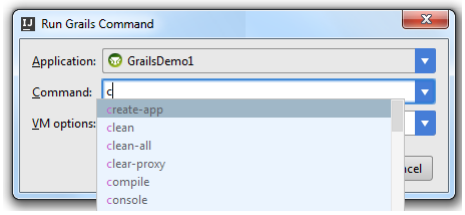
1. Select Grails or Griffon application module. You can do this in a number of ways:
   - In the Project tool window, select module of the corresponding type.
   - Bring forward Grails View or Griffon View tool window.
   - Open in the editor the desired file that pertains to a Grails or Griffon application.

2. On the Tools menu, choose Grails | Run Grails Command or Griffon | Run Griffon Command:



   Alternatively, just press `Ctrl+Alt+G`.

3. In the dialog box that opens, start typing the target name, and press `Ctrl+Space` to show the list of matching targets:



4. Click OK, and view the output messages in the console.

Please note the following:

- The history drop-down list is available in the RunTarget dialog box. Use the arrow keys to browse through the recently executed targets.
- You can run or debug any target immediately from the Run/Debug configuration, if you enter an arbitrary command in the Command line field of the Run/Debug Configuration dialog.

Running and Debugging Grails Applications

In IntelliJ IDEA, you can launch Grails or Griffon applications using the regular procedures, with the dedicated Grails Application or Griffon Application run/debug configuration:

- Running
- Debugging

Note Note that IntelliJ IDEA enables you to debug `*.gsp` files: you can set breakpoints at the lines of the views, examine variables, and evaluate expressions.

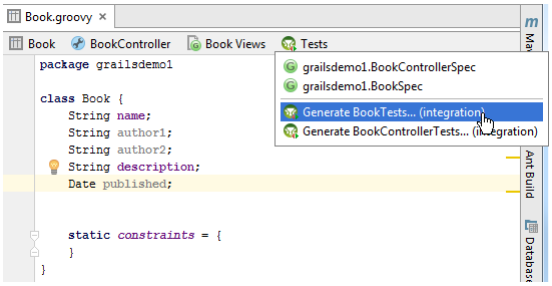
Besides that, you can execute Groovy scripts that exist in Grails or Griffon applications. Observe results in the Run and Debug tool windows.

Testing Grails Applications

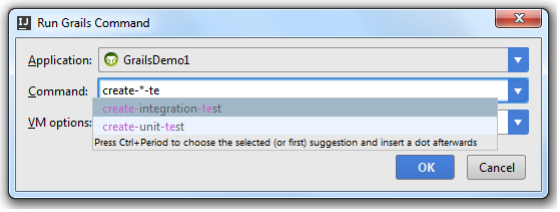
IntelliJ IDEA enables creating and running unit and integration Grails tests, and provides run configurations for each test type.

To create Grails test, do one of the following:

– Use scaffolding of a domain class you want to create a test for:

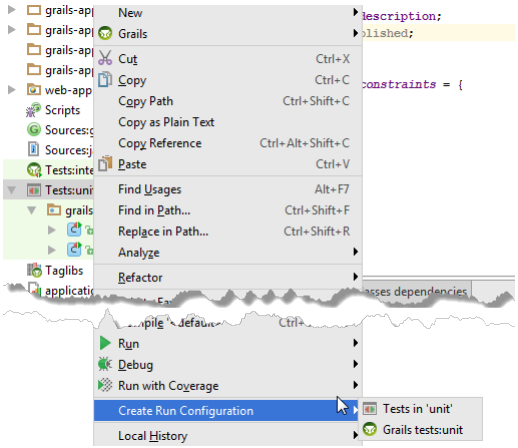


– Run Grails target : press `Ctrl+Alt+G`, and type the target name in the text field. Use code completion to narrow down the list of matching targets:



To create a Grails test run/debug configuration

1. In the Grails view , right-click one of the test directories, point to Create Run Configuration on the context menu, and then choose one of the suggested options:
   – Tests in...
   – Grails tests:...



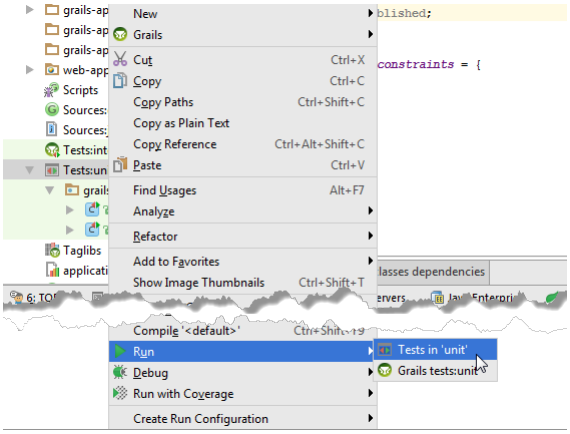
2. In the Run/Debug Configuration dialog that opens, specify the run/debug configuration settings:test type, Grails scripts etc.

> **Note** You can specify several test names separated with spaces, all these tests will run in a single run configuration.

To run a Grails test

1. In the Grails view , select the desired test class.

2. In the menu that opens, select Run and the desired configuration type:



The selected run/debug configuration is executed in the Run tool window.

Note Please note that for Grails versions 2.3.0 and later, you need to use Grails plugin repository to find and install Grails plugins. The IntelliJ IDEA plugin manager is available only for earlier Grails versions.

You can extend the functionality of your Grails application with the Grails plugins downloaded from the Grails repository. Once downloaded, the plugins reside under the `plugins` directory of your Grails application. Each plugin has the same directory structure as the entire Grails application. The source directories of the installed plugins are included in the source routes and are marked in the module tree view with the icon.

For the sake of better communication with the Grails repository, IntelliJ IDEA provides the plugin manager that enables viewing, installing and uninstalling plugins.

Note The first communication with the Grails repository from a Grails application can take time.

To install or uninstall the Grails plugins for Grails version 2.3.0 and later

- In your project, open `BuildConfig.groovy` file.
- In `grails.project.dependency.resolution.plugins` section of the code, enter `compile ":pluginName:pluginVersion"` of the plugin you want to install.

  If you need to remove the plugin, simply delete the specified line.

To open the Grails plugin manager, do one of the following

- On the main menu choose Tools | Grails | Plugins.
- On the context menu of the Project view, choose Grails | Plugins.

To view the available plugins

- Open the Grails plugin manager, click the refresh button if necessary, and view the list of plugins in the Grails plugins dialog box:

Tip You can also view the list of plugins by running the `list-plugins` Grails target. To do that, press `Ctrl+Alt+G`, and type `list-plugins` in the Run Grails Target dialog box.

To install or uninstall Grails plugins

1. Open the Grails plugin manager. In the Enable column, select the check boxes of the plugins to be installed, and clear the check boxes of the plugins to be uninstalled. Then click Apply Changes.
2. In the Install/Uninstall Grails plugins dialog box, review the list of plugins. If necessary, select the plugin version. Click OK.

Tip You can also install plugins using the `install-plugin` Grails target. To do that, press `Ctrl+Alt+G`, and in the Run Grails Target dialog box type `install-plugin <plugin name>`.

Dynamic Finders

Grails integration allows performing dynamic queries for the domain class instances. So doing, code completion makes it possible to combine different queries based on the fields of the domain classes.

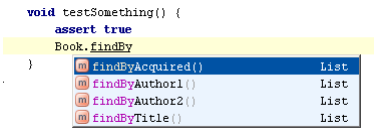
To create a dynamic query

1. In a Grails domain class, declare fields to define mappings. For example, in the domain class `Book.groovy` there are four fields:

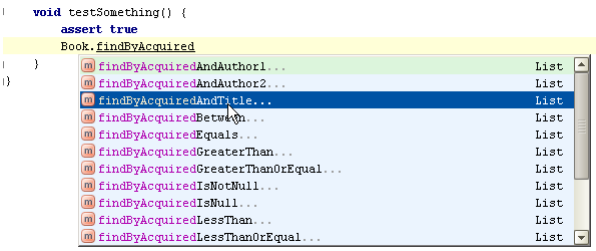
```
Date acquired
                    String title
                    String author1
                    String author2
```

2. In a Grails view, controller, or test class, create a method. For example, in the `BookTests.groovy`, create method `testSomething()`.

3. In the method body, reference a domain class to be queried, and start typing the query. Press `Ctrl+Space`:

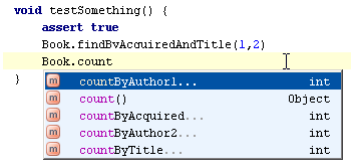


4. Press `Ctrl+Space` once more, and select the desired condition from the suggestion list:



Repeat code completion to concatenate as many search conditions as required.

The same procedure applies to the count function:



Griffon

Griffon support is available in the Community and in the Ultimate editions of IntelliJ IDEA.

In this section:

- Griffon
  - Prerequisites
  - Griffon features in IntelliJ IDEA
  - Griffon changes in the IntelliJ IDEA UI
- Creating a Griffon Application Module

Prerequisites

Make sure that the desired SDK is downloaded and installed on your computer, and the libraries are properly configured.

Also, make sure that the Groovy plugin is enabled in IntelliJ IDEA. The plugin is bundled with IntelliJ IDEA and is activated by default. If the plugin is not enabled, enable the plugin .

Griffon Features in IntelliJ IDEA

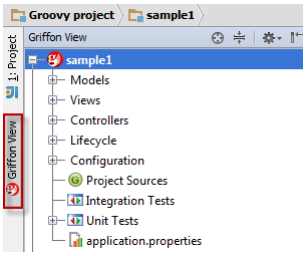
Griffon support in IntelliJ IDEA includes the following features:

- Automated way of creating Griffon Applications that provides generation of the specific structure and artifacts.
- Automatic enabling Groovy support in the Griffon Application modules, allowing creation of the Groovy classes, interfaces and scripts.
- Execution of the targets .
- ability to generate tests .
- Code completion .
- Dedicated run/debug configuration .

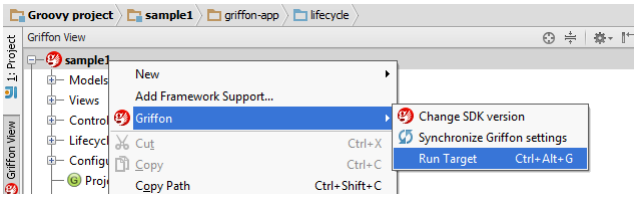
Griffon Changes in the IntelliJ IDEA UI

Once IntelliJ IDEA recognizes the project or module as a Griffon application, it introduces the following changes to the UI:

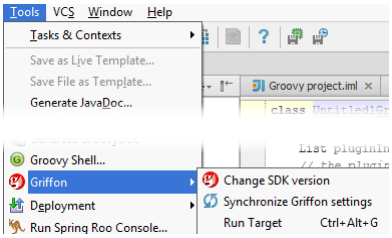
- Griffon View tool window is added, and the tool window button appears in the left tool windows bar:



- Griffon node is added to the context menu of the Griffon modules in the Griffon view:



- Griffon node is added to the Tools menu:



Creating a Griffon Application Module

You can create Griffon Application modules. IntelliJ IDEA generates the necessary infrastructure. You can view module structure in the Griffon tool window, and use framework-specific commands.

To create a Griffon Application module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Griffon . In the right-hand part of the page, specify the JDK that you are going to use and your local Griffon installation. (Griffon in IntelliJ IDEA is represented by a library .) If you need, specify additional libraries and frameworks.
   Click Next.

3. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Groovy

In this section:

- Groovy
  - Prerequisites
  - Groovy support
- Getting Started with Groovy
- Coding Assistance in Groovy
- Groovy Procedures
- Groovy-Specific Refactorings
- Launching Groovy Interactive Console

Prerequisites

Make sure that the desired SDK is downloaded and installed on your computer, and the libraries are properly configured.

Also, make sure that the Groovy plugin is enabled in IntelliJ IDEA. The plugin is bundled with IntelliJ IDEA and is activated by default. If the plugin is not enabled, enable the plugin .

Groovy support

IntelliJ IDEA enables you to use Groovy language and Groovy-based frameworks.

IntelliJ IDEA supports Groovy versions up to 2.4.

Groovy files are marked with icon.

Groovy support includes:

- Ability to add Groovy framework to the Java module dependency.
- Ability to create an extension module to support custom extension methods.
- Coding assistance .
- Common and specific refactorings.
- Numerous ways to navigate through the source code, among them:
  - Navigating with Structure View .
  - Navigate | Declaration (`Ctrl+B`).
  - Navigate | Implementation (`Ctrl+Alt+B`) from overridden method / subclassed class.
- Advanced facilities to search through the source code .
- Viewing reference information.
- Possibility to generate documentation, created according to GroovyDoc syntax.
- Possibility to compile mixed Groovy and Java code.
- Running and debugging .
- Possibility to create , and perform tests.

Getting Started with Groovy

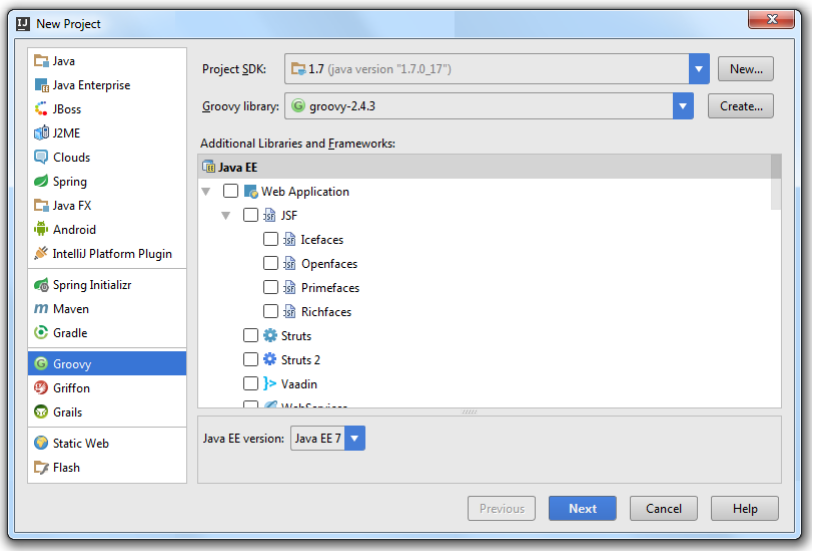
IntelliJ IDEA lets you create and run Groovy applications.

Before you start working with Groovy, make sure that the Groovy plugin is enabled in IntelliJ IDEA.

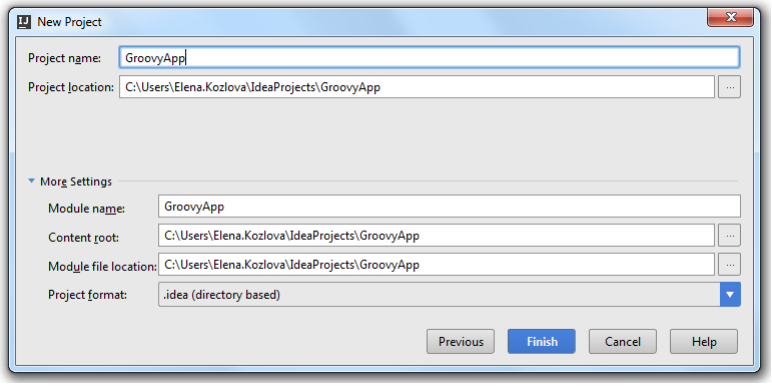
- Creating Groovy Project
- Adding Frameworks to Existing Groovy Project
- Creating Groovy Class
- Creating Groovy Script
- Running Groovy Application

Creating Groovy Project

1. Open Project Wizard , in the left-hand pane select Groovy.
2. In the right-hand pane, specify the following settings:
   - Project SDK - specify your project SDK.
   - Groovy library - specify your Groovy SDK or click Create to choose one from the list that opens.
   - Additional libraries and frameworks - specify additional libraries and frameworks that Groovy supports.
   - Java EE version - select the appropriate Java EE version.

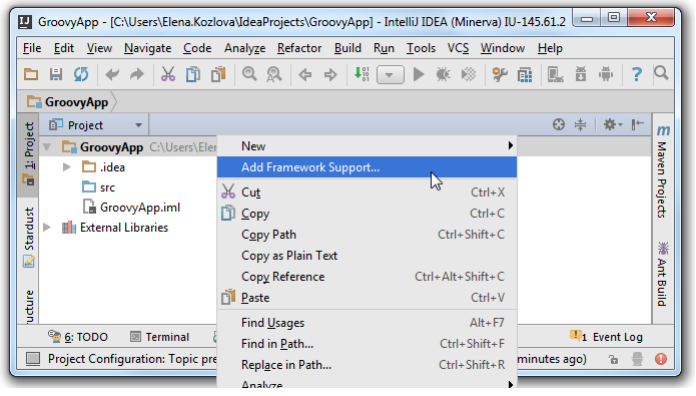


3. Click Next.
4. Specify your project information and click Finish.

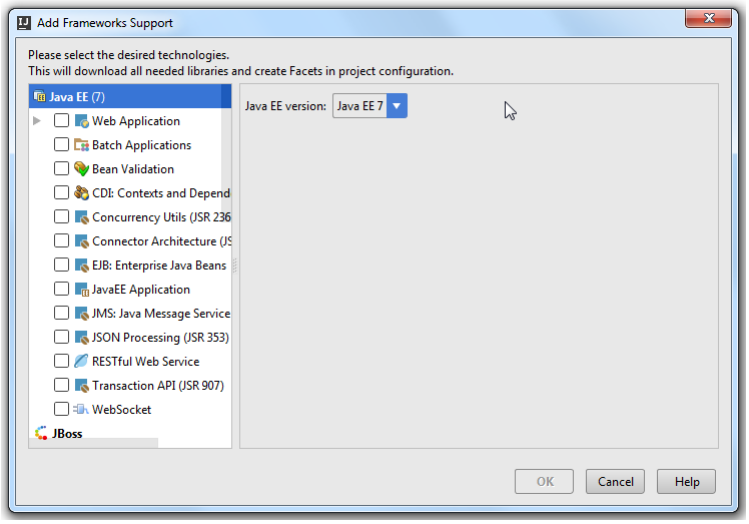


Adding Frameworks to Existing Groovy Project

1. In the Project tool window, right-click the project directory and from the drop-down list select Add Framework Support.

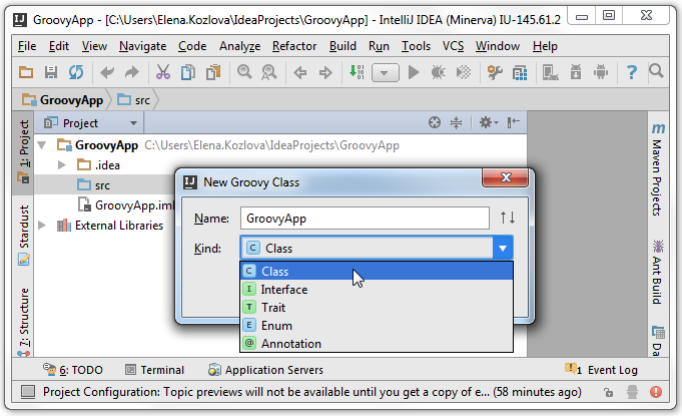


2. In the Add Frameworks Support dialog, select a framework and click OK.

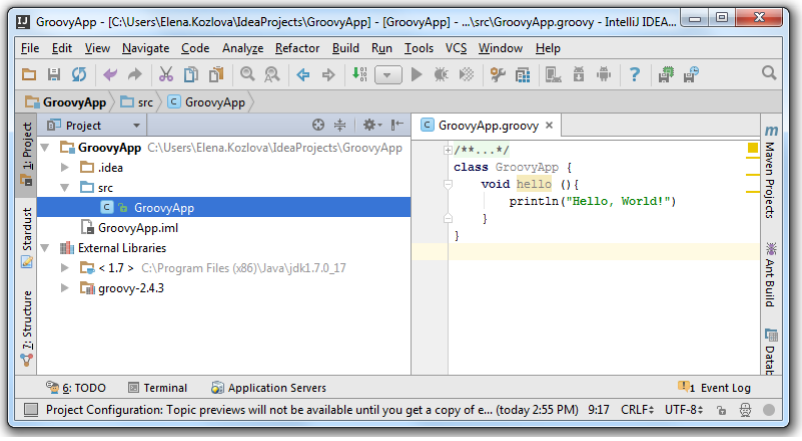


Creating Groovy Class

1. Press `Ctrl+N` or in the Project tool window right-click on the directory and select New | Groovy Class
2. In the New Groovy Class dialog, in the Name field, specify a name of the class. In the Kind field, choose between class, interface, trait, enum or annotation.

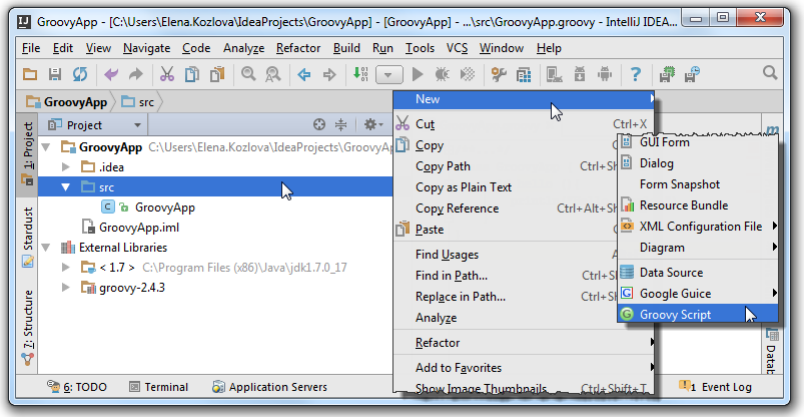


3. Now you can enter your code.

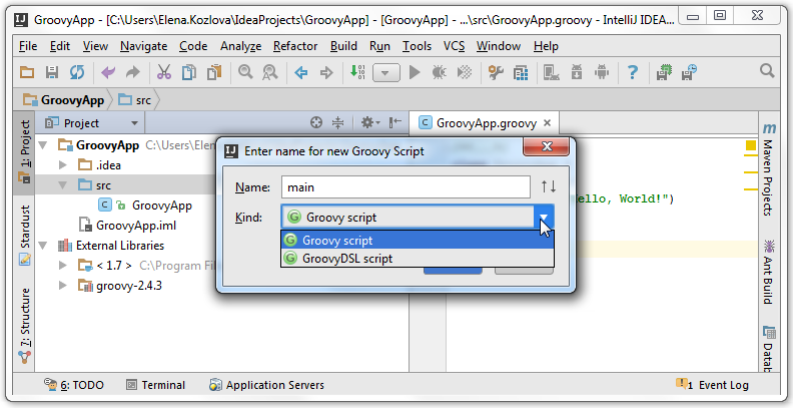


Creating Groovy Script

1. Press `Ctrl+N` or in the Project tool window right-click on the directory and select New | Groovy Script.



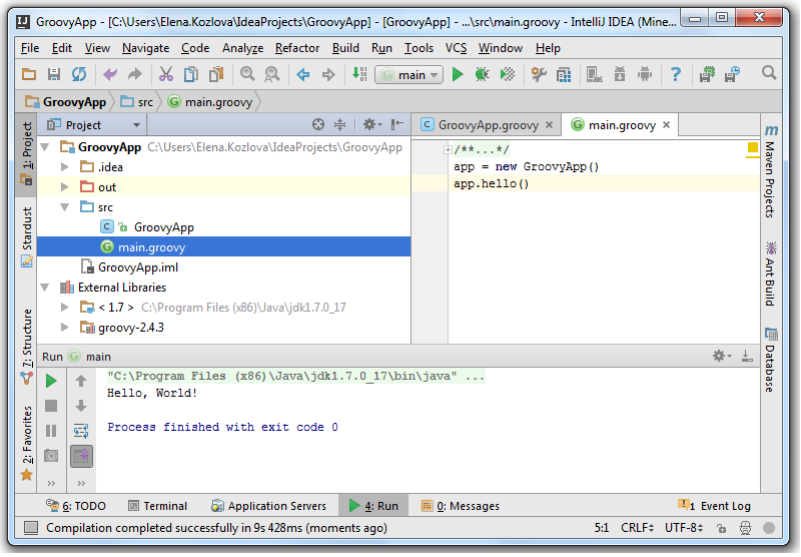
2. In the dialog that opens, in the Name field, enter the name of your Groovy script. In the Kind field, choose between Groovy script and GroovyDSL script.



3. IntelliJ IDEA creates a file with the specified name and `groovy` extension, and adds a node to the module tree view.

Running Groovy Application

1. Create a Run configuration using Run | Edit Configurations or run the active script automatically by pressing `Ctrl+Shift+F10`.



2. View the result in the Run tool window.

Coding Assistance in Groovy

IntelliJ IDEA supports the following coding assistance for Groovy:

- Code completion for keywords, labels, variables, parameters and functions.
- Chained expression completion .
- Error and syntax highlighting.
- Code formatting and folding .

  For example, all closures that have their parameters described in the separate line are formatted the following way:

```
def cl = {
                    param1, param2 ->
                    body()
                    }
```

- Numerous code inspections and quick-fixes .
- Ability to check what generics to specify for a method call or for a type variable using Parameter info (Ctrl+P).
- Code generation
  - Inserting, expanding, and generating code blocks using live templates .
  - Creating various applications elements via intention actions .
  - Possibility to create line and block comments (`Ctrl+Slash` / `Ctrl+Shift+Slash`).
- Ability to add annotations . For example, @Builder annotation. IntelliJ IDEA supports all Strategies from the standard library as well as code completion and navigation.

```
import groovy.transform.builder.Builder
@Builder
class GroovyApp {
    //You can enter fields for the class
}
```

- Expression Type action for Groovy expressions. You can use `Ctrl+Shift+P` to see the expression type without clicking the mouse.

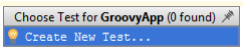
Groovy Procedures

This section covers Groovy-specific procedures:

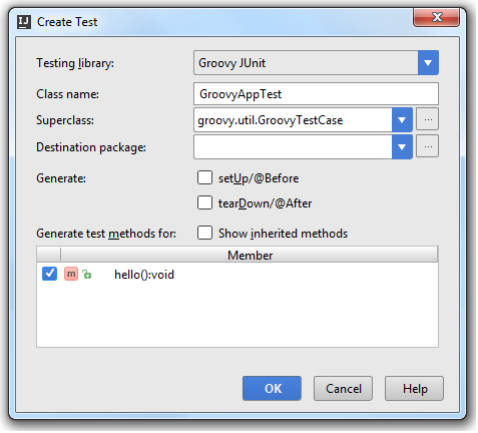
- Creating Groovy Tests and Navigating to Tests
- Generating Groovy Documentation
- Running and Debugging Groovy Scripts
- Working with Lists and Maps

To create a test for a Groovy class

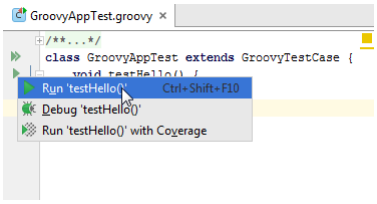
1. Do one of the following:
   - Use intention action Create test.
   - Press `Ctrl+Shift+T`, and choose Create New Test.



2. In the Create Test dialog box, specify the required information, as described in the section Creating Tests .



3. In the editor of the created test file, in the left gutter, click on the ▶ icon to execute run/debug configuration.



You can also use a context menu to run or debug your test.

Note You can navigate between Groovy tests and test subjects (Navigate | Test / Test Subject). If a test class doesn't yet exist, IntelliJ IDEA suggests to create one.

Generating Groovy Documentation

IntelliJ IDEA provides a front-end to the standard GroovyDoc utility for generating GroovyDoc reference for your Groovy-enabled project. This feature is available when the editor has the focus.

To generate Groovy project documentation

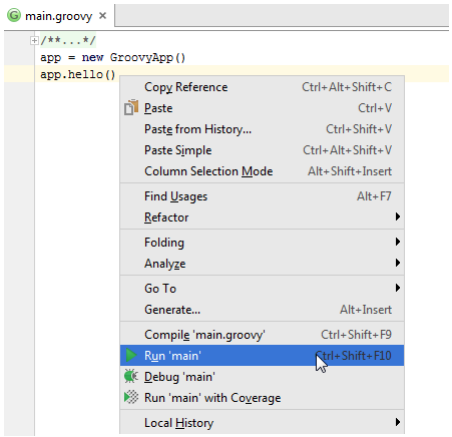
1. On the main menu, choose Tools | Generate GroovyDoc. Generate Groovy Documentation dialog box opens.
2. In the Generate GroovyDoc dialog, specify the input and output directories, and the visibility level. Refer to Generate Groovy Documentation for description of controls.
3. Click Start.

Running and Debugging Groovy Scripts

In IntelliJ IDEA, you can run and debug Groovy scripts using the regular procedures:

- Running
- Debugging

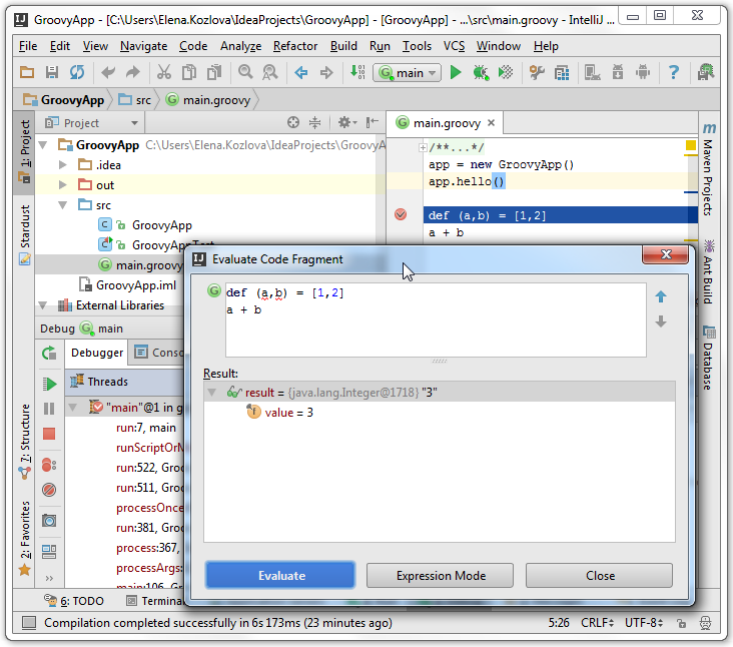
Besides executing a Groovy script with the permanent run/debug configuration, you can use a temporary one, which is available at the context menu of a script:



Observe results in the Run and Debug tool windows. Note that in Groovy-enabled modules, the debugger is aware of the Groovy syntax, and enables handy evaluation of expressions.

To evaluate Groovy expression

1. Launch the debugger session, as described in the section Debugging .
2. When you reach a breakpoint, where you want to evaluate expression, press `Alt+F8`.
3. In the Code Fragment Evaluation dialog box, select Groovy from the Language drop-down list, type the desired expression, and click Evaluate:



Working with Lists and Maps

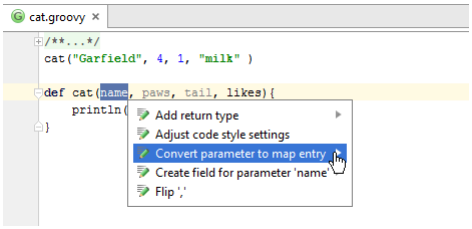
In this section you can find some examples of working with lists and maps in IntelliJ IDEA:

- To convert a parameter of a function to a map entry
- To convert Groovy map to a class instance
- To produce a list or a map
- To inline a list or a map

To convert a parameter of a function to a map entry

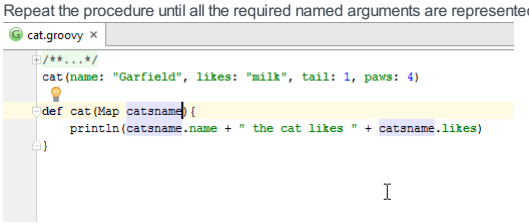
If a function has a lengthy list of named arguments, you can reduce it by representing parameters as map entries. For this purpose, IntelliJ IDEA suggests the Convert parameter to map entry intention action.

1. Place the caret at a parameter you want to convert, and press `Alt+Enter`:



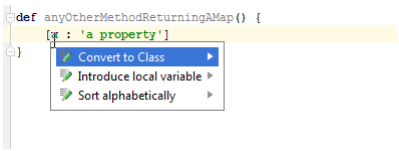
2. On the context menu, choose Convert parameter to map entry, and specify conversion parameters:

3. Repeat the procedure until all the required named arguments are represented as map entries. Note that all the subsequent conversions are done silently.



To convert Groovy map to a class instance

1. In a Groovy method, place the caret at a map to be converted, and press `Alt+Enter`:



2. On the context menu, choose Convert to class, and specify new class name and the package where the new class will be created. If such package doesn't yet exist, you can create one immediately,by clicking .

3. If necessary, choose to change return type of the method.

4. Click OK to apply changes and perform conversion.
   As a result, a class is created, with the fields corresponding to the keys of the original map.

To produce a list or a map

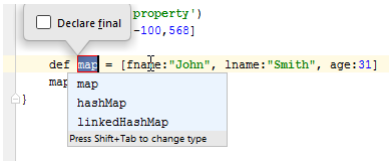
1. Type contents of a list or a map in the editor:

```
[55, 127,-9,-100,568]
```

or

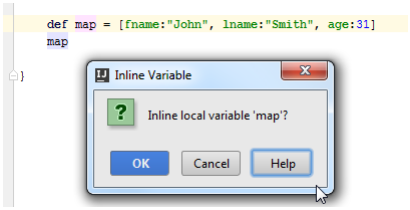
```
[fname:"John",lname:"Smith",age:31]
```

2. Place the caret somewhere inside the square brackets, and press `Ctrl+Alt+V`. The expression in brackets is converted to a list or map respectively.



To inline a list or a map

1. Place the caret on the list or map declaration, and press `Ctrl+Alt+N`. IntelliJ IDEA highlights the encountered usages:



2. Confirm inlining:

Groovy-Specific Refactorings

This section covers Groovy-specific refactoring procedures:

- Extracting Parameter in Groovy
- Extracting Method in Groovy

Extracting Parameter in Groovy

This section discusses the Extract Parameter refactoring for Groovy. This refactoring lets you perform the following actions:

- Creating a new method parameter from the selected expression within a method. All the usages of the method will be auto-updated.
- Adding a parameter to a closure. If there is a variable, associated with the closure, the calls to this variable are replaced with the corresponding expressions.

In this section:

- Examples
- To extract a parameter in Groovy

Examples

Before After

```
class Cat {
                    Cat cat = new Cat()
                    def makePestOfItself(){
                    print ("miaou!!!!!!!!")
                    }
                    def makeTroubles(){
                    if (makePest){
                    makePestOfItself()
                    }
                    }
                    }
```

```
class Cat {
                    Cat cat = new Cat()
                    def makePestOfItself(String warning){
                    print (warning)
                    }
                    def makeTroubles(){
                    if (makePest){
                    makePestOfItself("miaou!!!!!!!")
                    }
                    }
                    }
```

```
class Bar {
                    def foo = {
                    print 'H<caret
                    here>ello, world!'
                    }
                    }

                    new Bar().foo()
                    new Bar().foo.call()
```

```
class Bar {
                    def foo = { String
                    s ->
                    print s
                    }
                    }

                    new Bar().foo('Hello, world!')
                    new Bar().foo.call('Hello,
                    world!')
```

To extract a parameter in Groovy

1. In the editor, place the cursor within the expression to be replaced by a parameter.
2. Do one of the following:
   - Press `Ctrl+Alt+P`.
   - Choose Refactor | Extract | Parameter in the main menu.
   - Choose Refactor | Extract | Parameter from the context menu.
3. In the Extract Parameter dialog:
   1. Specify the parameter name in the Name field.
   2. Choose parameter type, and specify whether you want to declare the new parameter final, and create the overloading method.
   3. Click OK.

Extracting Method in Groovy

This section discusses Extract Method refactoring in Groovy.

This refactoring lets you perform the following actions:

- Extract method for a variable.
- Extract method for a list of variables.
- Extract method for one or several statements.

In this section:

- Examples
- To extract a method in Groovy

Examples

Before After

```
iii = 6
                    int kkk = 5
                    def vv = 6
                    def gg = 7

                    println (kkk + iii
                    + (vv +gg))
```

```
iii = 6
                    int kkk = 5
                    def vv = 6
                    def gg = 7

                    println(kkk + iii
                    + testMethod(vv,
                    gg))

                    private int testMethod(int vv, int gg) {
                    return vv + gg
                    }
```

```
def a = 5
                    print a
                    def b = 6
                    def c = 7
                    print a + b

                    print a + b +
                    c
```

```
def a = 5
                    int b, c
                    (b, c)
                    = testMethod(a)

                    private List testMethod(int a) {
                    print a
                    def b = 6;
                    def c = 7
                    print a + b
                    return [b, c]
                    }
                    print a+b+c
```

```
static def foo (int
                    i, int j,
                    int k){
                    def v
                    println(i + j
                    - k)
                    v = 42

                    if (i > 42) {
                    println("hello!")
                    } else {
                    return v + j
                    }
                    return 239
                    }
```

```
static def foo(int
                    i, int j,
                    int k) {
                    def v
                    println(i + j
                    - k)
                    v = 42
                    return testMethod(i,
                    v, j)
                    }

                    private static int testMethod(int i, int
                    v, int j)
                    {
                    if (i > 42) {
                    println("hello!")
                    } else {
                    return v + j
                    }
                    return 239
                    }
```

To extract a method in Groovy

1. In the editor, select a block of code to be transformed into a method or a function.

   Tip The code fragment to form the method does not necessarily have to be a set of statements. It may also be an expression used somewhere in the code.

2. On the main menu or on the context menu of the selection, choose Refactor | Extract | Method or press `Ctrl+Alt+M`.
3. In the Extract Method dialog box that opens, specify the name of the new method.
4. To return the value of a data type explicitly, select the Specify return type explicitly check box.
5. To return a keyword, select the Use explicit return statement check box.
6. In the Parameters area, do the following:
   - Specify the variables to be passed as method parameters, by selecting/clearing the corresponding check boxes; if a parameter is disabled, a local

variable of the corresponding type, with the initial value ... will be created in the extracted method, so that you will have to enter the initializer with an appropriate value manually.

- Rename the desired parameters, by double-clicking the corresponding parameter lines and entering new names.

7. Check the result in the Signature Preview pane and click OK to create the method. The selected code fragment will be replaced with a method call. Additionally, IntelliJ IDEA will propose to replace any similar code fragments found within the current class.

Launching Groovy Interactive Console

IntelliJ IDEA lets you launch an interactive Groovy console in any project. You can use the console as a temporary file to write and evaluate your code. If dependencies in your project contain a Groovy library then the specified Groovy library is used to launch the Groovy console. If the dependencies do not contain a Groovy library then the bundled Groovy library of the Groovy version 2.3.9 will be used.

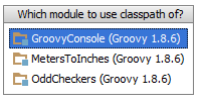
Note You can access the Groovy console in any Java project (Tools | Groovy Console).

In this section:

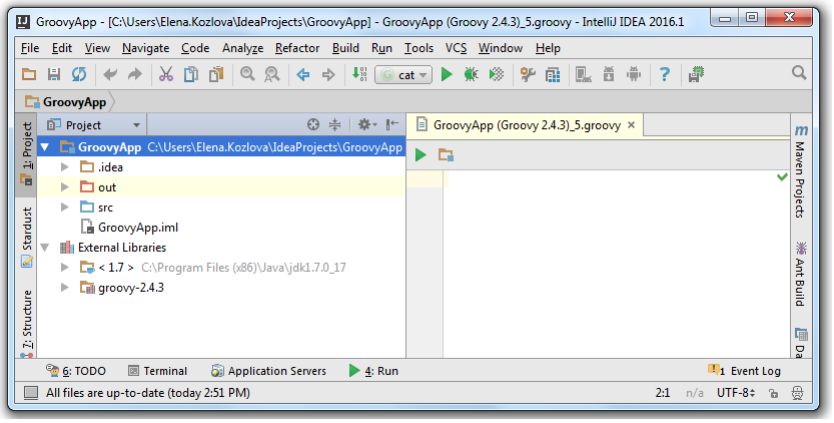
– Launching the Groovy interactive console
– Working in Groovy interactive console

To launch Groovy console

1. On the main menu, choose Tools | Groovy Console.
2. If your project consists of two modules or more, choose the module to use the classpath of:

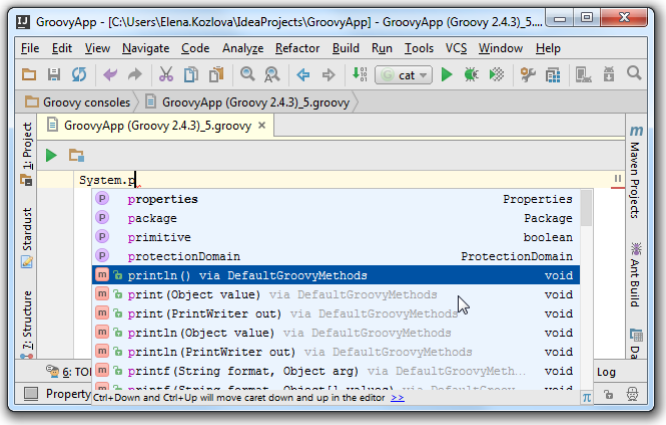


The Groovy console starts in a separate tab in the editor:

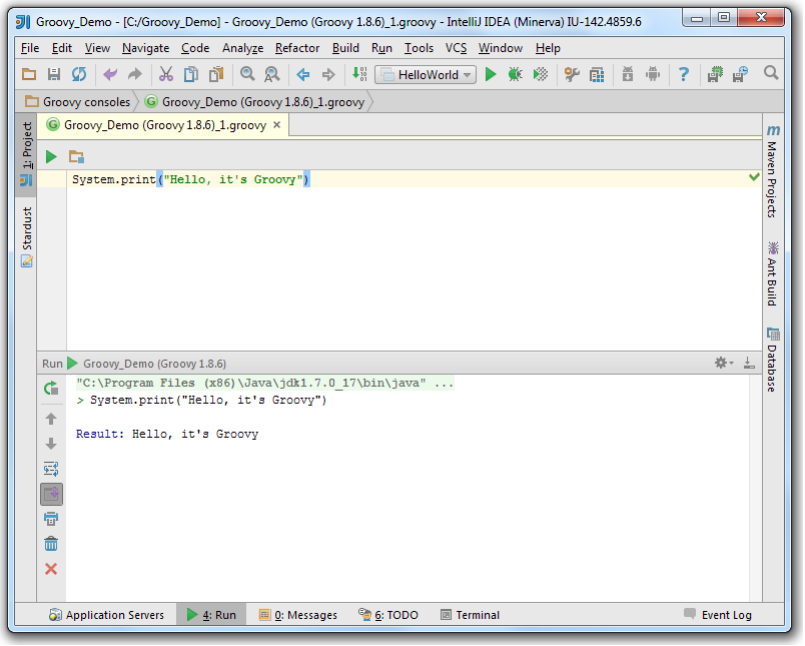


To use Groovy interactive console

1. Type code in the console after the prompt character, or just paste from a different file.
   Note that coding assistance is available, as you type (code completion and error highlighting):



2. Click ▶, or press `Ctrl+Enter` to execute the entered code.



3. View the results in the Run Tool Window for Groovy Console .
4. Use up and down arrow keys to navigate through the results' history. Depth of the history is defined by the Console commands history size in the Editor settings.

GWT

IntelliJ IDEA comes bundled with the GWT Support plugin that provides native support for developing Web applications based on the Google Web Toolkit (GWT).

IntelliJ IDEA GWT support, that extends to the version 2.7, includes the following features:

- GWT-aware coding assistance, such as refactoring , code completion, code inspections with quick-fixes in the following areas:
  - client-side Java code
  - remote services
  - JavaScript methods in Java code
  - enhanced syntax in *.css files
  - UiBinder *.ui.xml files
- Navigation to implementation , `Ctrl+Alt+B` .
- Intention Actions that let create various application elements.
- Code blocks, live templates , and file templates .
- Integration with GWT compiler.
- GWT-ready internationalization (i18n).
- Automatic creation of GWT components : modules, entry points, remote services and serializable classes.
- Support of GWT specific run configurations for running and debugging GWT applications directly from IntelliJ IDEA.
- Basic Vaadin support, implemented by the Vaadin Support plugin that comes bundled with IntelliJ IDEA.

Enabling GWT Support

IntelliJ IDEA supports the use of Google Web Toolkit through the dedicated GWT Facet . Before you enable GWT support, make sure GWT SDK is downloaded and installed on your computer.

When you have downloaded and installed GWT SDK, there are several ways to enable the GWT support, depending on your context:

- Create a module with a dedicated GWT facet.
- Attach a GWT facet to an existing module.

To have IntelliJ IDEA pick the path to the GWT installation folder every time you enable GWT support for a module, specify this path in the default GWT facet settings.

To configure default path to the GWT

1. Open the Project Structure dialog (`Ctrl+Shift+Alt+S`).
2. Go to Facets, and select GWT.
3. In the Defaults tab, specify the path to the GWT installation folder. Click OK.

Creating a Module with a GWT Facet

When you create a module with a dedicated GWT facet , IntelliJ IDEA configures the module and adds all the necessary libraries automatically.

To create a module with a GWT facet

Depending on the situation, you can choose to create a new project or to add a module to an existing project.

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Java Enterprise . In the right-hand part of the page, specify the JDK to be used and select the Java EE version to be supported.

3. Under Additional Libraries and Frameworks, select the Google Web Toolkit check box.
   In the GWT SDK field, specify the GWT SDK location. (To download the latest version of the GWT SDK, use the Download GWT link.)

   If you want a sample GWT application to be created, select the Create sample application check box and specify the application name in the field underneath.

   Click Next.

4. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Adding a GWT Facet to a Module

When you add a dedicated GWT facet to a module, IntelliJ IDEA re-configures the module and adds all the necessary libraries automatically.

To add a GWT facet to an existing module

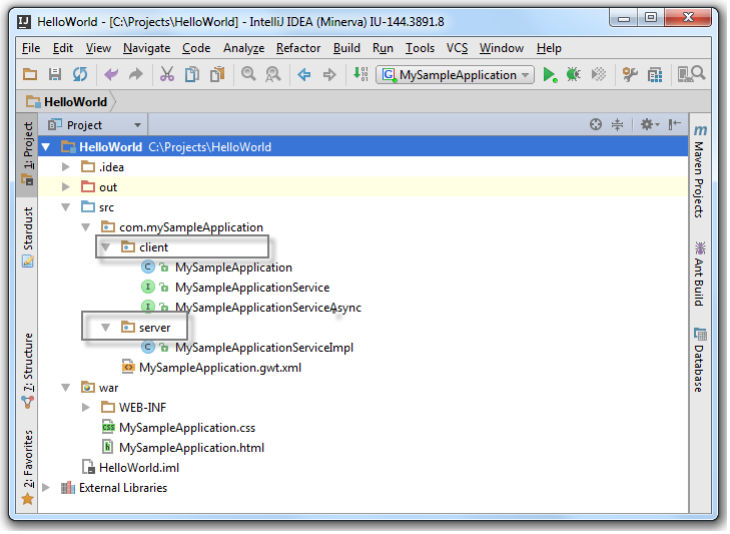
1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the module and select Add Framework Support.
3. In the left-hand pane of the Add Frameworks Support dialog, select the Google Web Toolkit check box.
4. In the GWT SDK field, specify the GWT SDK location. (To download the latest version of the GWT SDK, use the Download GWT link.)
5. If you want a sample GWT application to be created, select the Create sample application check box and specify the application name in the field underneath.
6. Click OK in the Add Frameworks Support dialog.

Developing GWT Components

When the GWT support is enabled , you can start developing GWT application components.

GWT Package Structure

The standard GWT package layout facilitates differentiating the client-side code from the server-side code. The image below illustrates the structure of a standard GWT package.



- Client - this directory contains the client-side source files and subpackages.
- Public - this directory contains various static resources that can be served publicly. By default, this directory is not created in the project.
- Server - this directory contains the server-side code and subpackages.
- GWT Module XML descriptor.

GWT Module

Individual units of a GWT configuration are XML files called modules. A module bundles all the configuration settings that your GWT project needs, namely:

- Inherited modules.
- An entry point application class name; these are optional, although any module referred to in HTML must have at least one entry-point class specified.
- Source path entries.
- Public path entries.
- Deferred binding rules, including property providers and class generators.

The GWT Module XML descriptor (5) should reside in the root package of a standard project layout. IntelliJ IDEA can simply generate a GWT Module with the corresponding project structure for you.

Entry Point

A module entry-point is any class that is assignable to `EntryPoint` and that can be constructed without parameters. When a module is loaded, every entry point class is instantiated and its `EntryPoint.onModuleLoad()` method is called.

Tip To get more familiar with the GWT application structure, have IntelliJ IDEA generate a GWT Sample Application for you.

In this part:

- Creating a GWT Module
- Creating an Entry Point
- Creating a Remote Service
- Creating a GWT UiBinder
- Creating GWT UiRenderer and Ui.XML File
- Creating Event and Event Handler Classes
- Creating GWT Serializable Class

Creating a GWT Module

With IntelliJ IDEA, you do not need to remember the required GWT module structure because a GWT module will be generated along with the standard package layout. IntelliJ IDEA also enables code completion for you by adding the relevant GWT DOCTYPE to the `.xml` descriptor of a new module automatically.

To generate a GWT module

1. In the Project view, right-click the package where you want to generate a GWT module. If you already have a GWT module, select any package outside the existing GWT module, to generate a new one.
2. From the context menu, select New | Google Web Toolkit | GWT Module.
3. Specify the name for the module and directory where html and css files should be created, then click OK. IntelliJ IDEA creates an `xml` GWT Module descriptor and generates the correct package layout:
   - The client part with an entry point in it.
   - The server part.
   - A public folder with an `html` page and a `css`.

Creating an Entry Point

A GWT module entry point is any class that can be assigned to `EntryPoint` and can be constructed without parameters. When a module is loaded, every entry point class is instantiated and its `EntryPoint.onModuleLoad()` method is called.

To generate an entry point

1. In the Project view, right-click the client package.
2. From the context menu, select New | Google Web Toolkit | GWT Entry point.
3. Specify the name of the entry point and click OK.

Creating a Remote Service

Developing a GWT service requires creating synchronous and asynchronous interfaces as well as server implementation with the correct names and structure. Also they should be registered in the `xml` module descriptor. IntelliJ IDEA can generate all the required parts automatically.

To generate a remote service

1. In the Project view, right-click the client package.
2. From the context menu, select New | Google Web Toolkit | GWT Remote service.
3. Specify the name for the service and click OK.

Creating a GWT UiBinder

IntelliJ IDEA can generate a GWT UiBinder and a `ui.xml` file for you. While defining the declarative layout in the `ui.xml`, various types of coding assistance, such as code completion and generation of import statements, are at your disposal.

IntelliJ IDEA supports integration between the declarative layout in the `ui.xml` and the corresponding Java class, including Finding usages.

If you are using the `ClientBundle` interface, IntelliJ IDEA marks its methods with the icon in the left gutter area. Click this icon to navigate to the corresponding resource.

Warning! A GWT UiBinder can be created in the client part only.

To generate a GWT UiBinder

1. In the Project view, right-click the client package.
2. From the context menu, select New | Google Web Toolkit | GWT UiBinder and ui.xml file.
3. In the Create GWT UiBinder and ui.xml file dialog box that opens, specify the name of the UiBinder and the ui.xml file.
4. Specify the type of the root element. Select the relevant item from the drop-down list or click the Browse button and choose the relevant type in the Search dialog box that opens.

Creating GWT UiRenderer and ui.xml file

IntelliJ IDEA can generate a GWT UiRenderer and a `ui.xml` file for you. While defining the declarative layout in the `ui.xml`, various types of coding assistance, such as code completion and generation of import statements, are at your disposal.

IntelliJ IDEA supports integration between the declarative layout in the `ui.xml` and the corresponding Java class, including Finding usages.

If you are using the `ClientBundle` interface, IntelliJ IDEA marks its methods with the icon in the left gutter area. Click this icon to navigate to the corresponding resource.

Warning! A GWT UiRenderer can be created in the client part only.

Generating GWT UiRenderer

1. In the Project view, right-click the client package.
2. From the context menu, select New | Google Web Toolkit | GWT Renderer and ui.xml file.
3. In the Create New GWT Renderer and ui.xml file dialog box that opens, specify the name of the UiRenderer and the ui.xml file.

   Click OK.

Creating Event and Event Handler Classes

IntelliJ IDEA can generate stubs for GWT Event classes and GWT Event Handler interfaces.

Tip GWT Event classes and Event Handler interfaces can be created in the client part only.

To create a GWT event and event handler

1. In the Project view, right-click the client package.
2. From the context menu, select New | Google Web Toolkit | GWT Event and Handler classes.
3. In the Create GWT Event and Handler Classes dialog box that opens, specify the names of the event class, the event handler interface, and the abstract "callback" method for callers to override.
   Actually, you do not need to specify all the three names manually. As soon as you type the event class name in the Event class name text box, IntelliJ IDEA automatically completes the names of the handler interface and the "callback" method and displays the suggestions in the Handler class name and Handle method name text boxes respectively. This ensures that the generated stubs will be generated in accordance with GWT Google API libraries.
4. When you click OK, IntelliJ IDEA generates the stubs for the event class and event handler interface and opens them in the editor. Populate the generated stubs as necessary.

Creating GWT Serializable Class

IntelliJ IDEA can generate a serializable class for you, that will meet all the requirements:

- Implements `IsSerializable`.
- Has a `public` default (zero argument) constructor.

To generate a serializable class

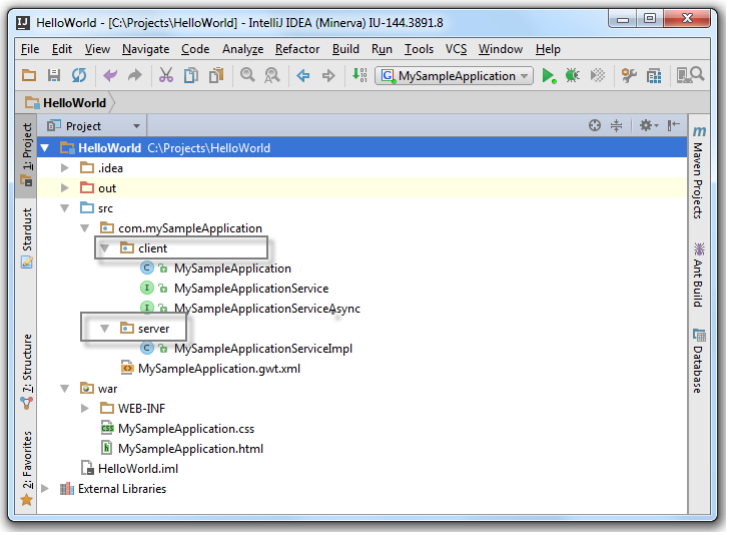
1. A serializable class can be created in the client part only.
2. All non-final, non-transient instance fields within the class must be serializable themselves.

1. In the Project view, right-click the client package.
2. From the context menu, select New | Google Web Toolkit | GWT Serializable class.
3. Specify the name of the class and click OK.

GWT Sample Application Overview

IntelliJ IDEA can generate a sample GWT application when you are creating a module with GWT support . This application helps you familiarize yourself with the GWT application structure. For a detailed description of the GWT fundamentals, please refer to http://code.google.com/webtoolkit/overview.html .

IntelliJ IDEA creates a simple HelloWorld application that meets all the GWT requirements.



As you can see, IntelliJ IDEA has generated a GWT Module (`MySampleApplication.gwt.xml`) in the root of the package. The corresponding package layout (client and server parts) is also generated automatically.

- `client` : contains the required entry point (`MySampleApplication` class) and remote service interfaces.
- `server` : contains the server implementation of the remote service.
  When creating a remote GWT service, IntelliJ IDEA automatically generates both synchronous and asynchronous interfaces (in the `client` part), creates the server implementation, and registers them in the GWT Module `.xml` , as required.

The HTML host file, that is used to run the application, is located in a separate Web facet.

IntelliJ IDEA also adds comments to the sample application code to make it clear to the maximum.

Analyzing GWT Compiled Output

IntelliJ IDEA helps you understand the compiled output of your GWT application by providing GWT Compile Reports . Generated reports are displayed in the browser.

To view the GWT compile output report

1. On the main menu, choose Tools | Generate GWT Compile Report.
2. In the Generate GWT Compile Report dialog box that opens, choose the module to view the output of from the GWT Module drop-down list.
3. Specify whether you want to view an existing report or a new one. Do one of the following:
   - To have a new report created, click the Generate button.
   - To view the previous generated report, click the View Report button.

   Tip IntelliJ IDEA informs you how long ago the last report was generated.

Opening a GWT Application in the Browser

You can specify whether you need the application to open in a browser or not. Moreover, you can choose a browser to use. This can be the default IntelliJ IDEA browser or any other browser of your choice.

To enable or disable opening a GWT application in the browser

1. Open Run/Debug Configuration for GWT .
2. Select Open in browser check box and from the drop-down list select a browser that you want to use for your application. Otherwise, clear the Open in browser check box.
   Note that even if you decide to run the application in the embedded GWT development mode without launching the browser, you can still specify the `.html` file from which you can start the application.

To configure the way to show a GWT application

1. Open Run/Debug Configuration for GWT .
2. Select the Open in browser check box and from the drop-down list, select the browser in which you want to open the application.
   To use the default IntelliJ IDEA browser, choose the Default item from the drop-down list.
3. To specify the starting page of the application, in the Start page field, select the `.html` file that implements this page.

IntelliLang

This a third-party documentation.

The IntelliLang plugin offers a number of features related to the use of custom languages in IntelliJ IDEA.

Prerequisites

Before you start working with language injections, make sure that the IntelliLang plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

In this part:

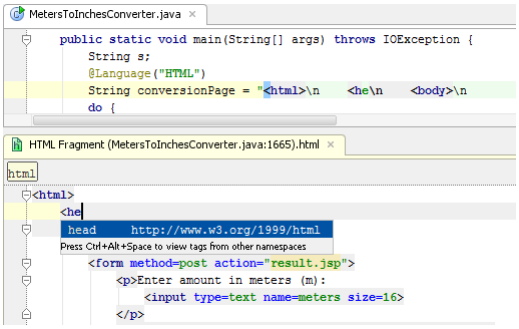
– Overview
– General Usage
– Code Inspections
– Quick Edit Language
– Usage Examples
– IntelliLang Configuration
– Configuring Libraries and Dependencies in Grails Applications

Overview

IntelliLang is a combination of three basic kinds of functionality that are meant to support the developer in dealing with certain tasks that relate to (custom) languages in IntelliJ IDEA:

- Language Injection: Editing the code that is embedded into String literals and XML fragments natively.

  You can open and modify an injected language code fragment in the editor . In this way you get all the necessary coding assistance, as if you were working with the source code in the corresponding language.



  To open an injected language code fragment in the editor, use the Edit <Language> Fragment intention action .

- Pattern Validation: Provides assistance in making sure that Strings being passed to and from methods match a particular regular expression
- Regular Expression Support: A custom language implementation for regular expressions

Language Injection

This makes use of the new possibilities of IntelliJ IDEA to treat String literals, XML text and attributes as fragments of an arbitrary language (called *Language Injection*). The plugin makes this newly introduced API readily available to everybody for their daily use through two very simple means: Either by using some Java annotations to mark String fields, local variables, method parameters and methods returning Strings as containing a certain language, or by just using a simple UI configuration. There is a set of annotations provided by the plugin, but the actual annotations are freely configurable to avoid any unwanted dependencies.

This enables the developer to get the benefit of a wide range of edit-time features, such as syntax error highlighting, completion, inspections and a lot more while editing fragments of e.g. JavaScript inside regular Java code or in XML files of custom schemas that IntelliJ IDEA usually doesn't know about.

Pattern Validation

Additionally, the plugin allows you to annotate Java elements of type String to have them checked for compliance with certain Regular Expressions. This can be useful for very simple *languages* where the developer needs to make sure that an expression conforms to a certain syntax, e.g. that a String is a legal Java identifier or a valid printf-like pattern used by `java.util.Formatter` .

This can both be validated on-the-fly while editing the code as well as during runtime (method parameters and return value only, like the @NotNull instrumentation of IntelliJ IDEA core) by instrumenting the compiled classes with assertions that match the value against the supplied pattern.

Regular Expression Support

This part of the plugin implements language-support for `java.util.regex.Pattern` and has been mainly created to support the IntelliLang plugin by adding support for the micro-language that is probably one of the most often used one inside Strings. It features complete support for the syntax of the SDK's regular expression implementation and adds some further features, such as

- Completion and validation for character property names (e.g. `\p{javaJavaIdentifierStart}`) which nobody can usually remember anyway
- Validation and navigation for the use of back-references (e.g. `\1`), e.g ctrl-b navigates to the capturing group the backref refers to.
- Intention Actions to simplify usages of repeated character occurrences, e.g. `a{0,1}` is offered to be converted to `a?`
- "Surround With" capturing/non-capturing group
- and more

General Usage

The plugin's usage is simple and straightforward, yet very flexible. Either add the provided set of annotations to your project and start using them, configure the plugin to use a custom set of annotations or simply use the UI configuration to make IntelliJ IDEA learn that e.g. the String argument of `Pattern.compile()` should be treated as a regular expression.

Using the Annotations

IntelliLang makes use of three base-annotations: `@Language`, `@Pattern` and `@Subst`

- `@Language` is responsible for the Language injection feature
- `@Pattern` is used to validate Strings against a certain regular expression pattern
- `@Subst` is used to substitute non-compile time constant expressions with a fixed value. This allows you to validate patterns and build the prefix/suffix of an injected language (see below) even for non-constant expressions that are known to contain certain kinds of values during runtime.

The annotations supplied with IntelliLang are located in the file `annotations.jar` which can be found in `/plugins/IntelliLang/lib`.

@Language

The `@Language` annotation can be used to annotate String fields, local variables, method parameters and methods returning Strings. This will cause String-literals that are assigned to a field/variable, passed as a parameter or are used as a method's return value to be interpreted as the specified language.

Additionally, there's the Language Mismatch inspection which checks for clashes between the *expected language* and the *actual language* when a field/variable is assigned or a value returned from a method.

The plugin supports *direct* and *indirect* annotations, i.e. you can either directly use the annotation like this:

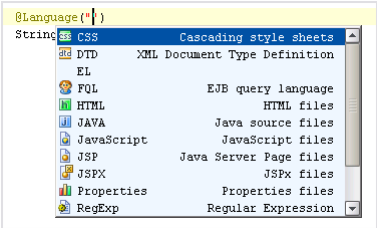
```
@Language("JavaScript")
                    String code = "var x = 1 + 2";
```

or annotate another annotation class like this:

```
@Language("XPath")
                    public @interface XPath { }
```

which can then simply be used to annotate elements as `@XPath`.

It's very easy to obtain the language-id that must be supplied as the annotation's value attribute: IntelliLang provides the list of available language via the regular code-completion action. Just select the appropriate language from the ctrl-space pop-up window:



@Pattern

The `@Pattern` annotation is responsible for marking Strings that have to comply with a certain regular expression and can be used in just the same way as the `@Language` annotation. That means, it's possible to create derived annotations, such as an `@Number` annotation that requires a String to consist of one or more digits:

```
@Pattern("\\d+")
                    public @interface Number { }
```

In fact, the predefined annotations already contain two of such derived annotations: The first one, `@PrintFormat`, matches the printf-like pattern used by `java.util.Formatter` and another one, `@Identifier`, describes a valid Java identifier. These are ready to use without having to specify any additional pattern.

@Subst

The `@Subst` annotation is used to substitute references that are not compile-time constant which enables the plugin to do Pattern Validation based on the assumption that the substituted value is compatible with the values that are expected during runtime. The plugin will complain if the value does not match the expected pattern.

It also helps to build a valid context of prefix/suffix (see next section) for the Language Injection feature. Consider this example:

```
@Subst("Tahoma")
                    final String font = new
                    JLabel().getFont().getName();

                    @Language("HTML")
                    String message = "<html><span
                    style='font: " + font + ";
                    font-size:smaller'>" + ... + "</span></html>";
```

Without substituting the value of the variable `font` with the value *Tahoma* (actually it could just be a single character here), the injected fragment would be syntactically

incorrect, causing the error *a term expected* to be displayed after the " `font:` " instruction.

Supplying Context: Prefix and Suffix

When annotating an element, it's possible to supply a prefix and a suffix that should be prepended/appended when the language fragment is being parsed. This can be used to supply context information, i.e. if the prefix for a JavaScript injection is "var someContextVariable;", IntelliJ IDEA will know that the variable *someContextVariable* is declared and will not warn about it being undeclared when it's used.

Apart from the manual ability to supply a prefix and suffix, IntelliLang dynamically determines those values from the context a String literal is being used in:

```
@Language(value =JavaScript, prefix = "function doSomethingElse(a){
                    }")
                    String code = "function doSomething()
                    {\n" +
                    "  var x = 1;\n" +
                    "  doSomethingElse(x);\n" +
                    "}";
```

In this example, the JavaScript language will be injected into each of the three String literals, and each one's prefix and suffix will be calculated from the preceding and following expressions so that the resulting text that's being parsed is valid JavaScript syntax:

- no "missing '}'" error will be displayed: The closing brace is part of the first literal's suffix.
- the variable `x` that is used in `doSomethingElse(a);` will be declared: Its declaration is part of the second literal's prefix
- the function `doSomethingElse()` will be known as well: It's defined in the statically supplied prefix

Warning! There are some issues with the IntelliJ IDEA Language Injection API that impose certain restrictions on the prefix/suffix of an injected language fragment. For instance, it's not allowed for a token of the language to span across the prefix/suffix of an element. This could e.g. happen if the prefix ends with a whitespace character and the fragment starts with whitespace. The plugin deals with this special situation in the way that it trims the prefix/suffix and inserts exactly one space character as a separator. However, this doesn't work if a space character is no token separator, which e.g. applies to JavaScript string literals. Such cases cannot be automatically dealt with and IntelliJ IDEA core will produce an assertion.

Note Even though the dynamic prefix/suffix calculation provides a proper context for the language fragment, some things may not work as expected. Most notably this are the refactoring (rename) and navigation functions. See also the known issues below.

Code Inspections

IntelliLang contains a set of inspections that validate the correct use of the supplied annotations (or custom configured ones). These inspections can be configured through the regular Settings | Errors configuration dialog.

Language Injection

The inspections in this category apply to the language injection feature related to the `@Language` annotation.

Unknown Language ID

This inspection provides validation for the use of non-existing Language-IDs. It flags usages of incorrect values of the `@Language` value attribute, such as `@Language("NonExistingID")`.

Language Mismatch

Validation for using references to elements that are annotated as containing different languages or are not annotated at all. The inspection offers a Quick-Fix to annotate such elements with the right annotation for the expected language.

Injection not applicable

This inspection checks whether an @Language or any derived annotation is used for anything other than elements of type `String` or `String[]`.

Pattern Validation

This category contains inspections about validating the use of the @Pattern or its derived annotations.

Validate Annotated Patterns

This inspection validates that expressions (String literals, as well as other compile-time constant or substituted expressions) match the pattern required by the `@Pattern` annotation. The inspection has an option to ignore non-constant expressions that contain non-substituted references and offers a Quick-Fix to add a substitution where applicable.

Pattern Annotation not applicable

Checks whether a pattern-validation annotation (`@Pattern` or derived ones) is valid to be applied to the annotated element. Only elements of type `String` may be annotated.

Non-annotated Method overrides @Pattern Method

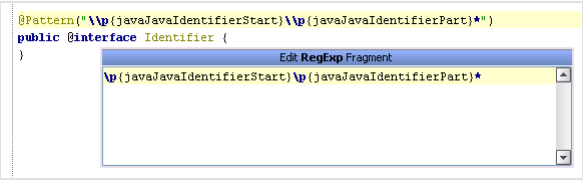
This inspection checks whether a method without any `@Pattern` or derived annotation overrides an annotated method from its base classes. This is not necessary for the error-highlighting inside the editor, however the runtime-check instrumentation doesn't pick up annotations from base-class methods.

A Quick-Fix is provided to add an annotation that matches the one from the base-class method. This ensures that the runtime-check instrumentation works correctly.

Quick Edit Language

This is an experimental feature that can be extremely helpful, especially for editing Regular Expression patterns inside String literals due to the *double escaping* requirement. For example, in a regular expression, a literal backslash character has to be written as a double backslash and each of them has to be escaped with another backslash when written inside a String literal.

The Quick Edit function that appears as an Intention Action for any injected language fragment displays a pop-up dialog that allows you to edit the string's value without the double escaping requirement. The dialog also shows the particular prefix/suffix of the fragment in a non-editable area.



The pop-up window can be dismissed by pressing Escape or clicking somewhere outside the pop-up window. Any changes made in the pop-up window are committed by pressing Ctrl-Enter.

Please note that the formatting of the non-editable prefix/suffix may differ from the actual value due to some problems. However, even though the formatting/alignment of the editable text may differ from the expected text, making changes to the text still works as expected.

Warning! Accepting Inspection Quick-Fixes inside the QuickEdit pop-up window may cause strange things to happen when the Quick-Fix attempts to open an Editor for the element being edited. The *Create Method* Quick-Fix of the JavaScript language is an example for that.

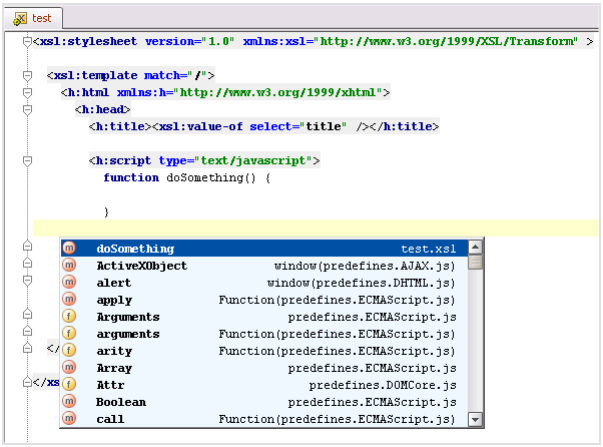
Usage Examples

Among the rather simple use-cases, like having detailed syntax checks for all kinds of "micro-languages" that are used e.g. in `Pattern.compile()`, `XPath.compile()` and so on, here are some less obvious, yet very useful examples how IntelliLang can leverage IntelliJ IDEA's support for a much better coding assistance.

The new scripting support in the upcoming Java 1.6 is another case where it will be important to get as much as possible edit-time assistance when script-code is constructed from Java code.

Extended JavaScript Support

When dealing with JavaScript that's not directly embedded inside an HTML page, IntelliJ IDEA usually just treats it as plain text. Consider the following example that creates an HTML page from an XSLT script. Without the JavaScript language being injected into the *script* tag with the XHTML namespace as shown in the screenshot below, this would be treated as plain text, with no further code assistance.



Support for JSP custom cags

With IntelliLang it's also possible to make the content and attributes of custom JSP tags being treated as another language. This can be useful e.g. for server-side scripting using JavaScript or any other Language implementation available for IntelliJ IDEA.

One thing that's important to know is that the taglib's URI which supplies a custom tag should be used as the namespace URI of the XML tag to inject a language into. The namespace-textfields contain a list of all known taglib URIs in the project.

Warning! Unfortunately, at the moment the support for refactoring and navigation inside JSP custom tags seems to be broken and attempting to use code completion may result in exceptions thrown in IntelliJ IDEA core. See also the known issues below.

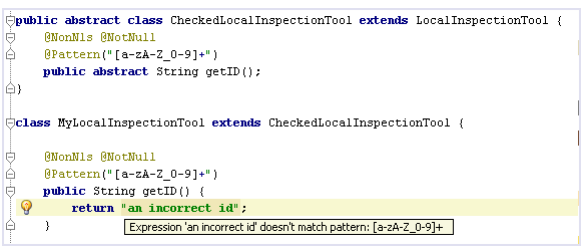
Pattern validation

Here's an obvious example right from IntelliJ IDEA's OpenAPI:

```
/** com.intellij.codeInspection.LocalInspectionTool

* @return descriptive name to be used in
suppress comments and annotations,
*         must consist of [a-zA-Z_0-9]+
*/
@NonNls @NotNull public String getID() {
return getShortName();
}
```

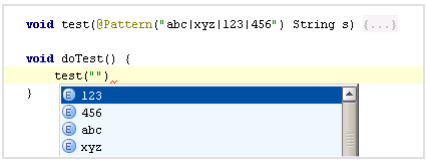
The contract of the method `getID()` is that it should only return strings that match the pattern "[a-zA-Z_0-9]+". The short note in the JavaDoc can be easily overlooked though because the contract isn't specified in an automatically verifiable way.

However, if this method were annotated as `@Pattern("[a-zA-Z_0-9]+")`, any attempt to return a string that doesn't match that pattern would be flagged in the editor:



Pattern completion

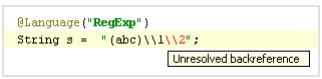
If a regular expression pattern represents an enumeration of different literal values, the plugin offers completion for those values:



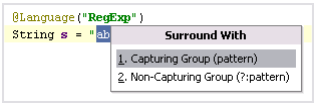
Regular Expression Editing

Here are some examples of the enhanced coding support for regular expression patterns:

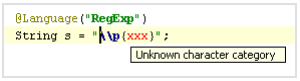
Backref validation



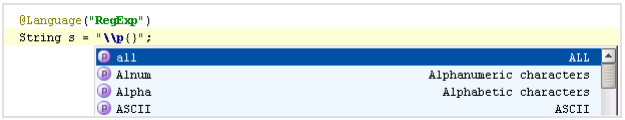
Surround with



Character category validation



Character category completion



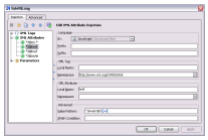
IntelliLang Configuration

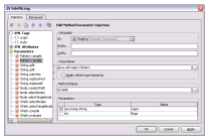
The configuration dialog provides the following options:

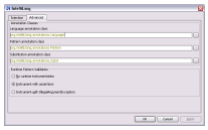
- Language injection for XML text
- Language injection for XML attributes
- Language injection for method parameters
- Advanced settings that let you change the names of the recognized base-annotations and the way how the pattern-validation is performed during runtime.

Screenshots

The screen shots below show some of the configuration options that can be tweaked and will be described in the following sections.







Language Injection

This tab allows you to configure the language injection feature for XML text, attributes and method parameters. Use the buttons in the toolbar or the context menu actions to add, remove, copy or import new entries. To add a new entry, the appropriate group has to be selected first.

Reordering the entries is possible with the Move Up and Move Down buttons. This can be important to define the precedence of different XML-injection entries. If an entry matches, no more injections are applied unless the injection specifies a value pattern .

XML Text

After adding a new XML text injection, select the ID of the Language to inject and optionally specify the prefix/suffix that make up the injection's context.

In the *XML Tag* pane, specify the local name (i.e. the name *without* any namespace prefix) and the namespace URI of the XML tag surrounding the text that should be treated as the selected language. The *name* field should not be empty, however the *namespace* field is optional.

The *Local Name* field takes a regular expression which makes it is possible to specify e.g. multiple tag names (name1|name2), case-insensitive names (e.g. (?i)tagname matches *tagname* as well as *TagName*), etc. Be sure not to enter any whitespace characters as they would be significant for the match.

XML Attributes

This is similar to the XML text configuration. However, it is possible to leave the *name* of the *XML Tag* empty which means that the configuration will apply to any attribute that matches the configured name, regardless of its containing XML tag.

The attribute's name takes the local name of the attribute and is also specified as a regular expression. This can be e.g. used to match e.g. HTML event handler attributes by specifying the name "on.*". The attribute name may also be empty (unless the tag name is empty as well) which means that the configuration applies to all attributes of the containing tag.

Advanced XML Options

The *Advanced* pane for XML Text and Attributes allows an even more fine-grained control over the injection process.

Value Pattern

This field takes a regular expression that determines what part of the XML text's or attribute's value the language should be injected into. This can be used to inject the language only into values that match a certain pattern or to inject it into multiple parts that match the pattern. This is done by using the first capturing group of the pattern as the target for the injection.

Examples:

`[$#]\{(.*?)\}`
This matches the pattern used by the JSP/JSF Expression Language.

`^javascript:(.*)`
This matches the *javascript*-protocol that can be used in hyperlink-hrefs to execute JS code.

XPath Condition

This field takes an XPath expression that can be used to address the injection-target more precisely than just by supplying its name and namespace URI. The context the expression is evaluated in is the surrounding XML tag for XML Text injection and the attribute itself for XML Attribute injection.

Example:

`lower-case(@type)='text/javascript'`
This limits the injection to tags whose type attribute contains the value "text/javascript".

It's possible to use the XPath extension functions that are provided by the Jaxen XPath engine, e.g. lower-case(). Also, there are three additional functions that can be used to determine the current file's name, extension and file type: file-name(), file-ext() and file-type(). You can also use normal code-completion to get a list of available functions.

Warning! For performance reasons, it's recommended to keep these expressions as simple as possible. Especially expressions that cause the whole document to be scanned, such as //foo/bar might cause performance problems with large files.

Method Parameters

This is a possibility to make use of IntelliLang's features, if, for any reason, the injection annotations cannot be used. This mainly applies to configuring 3rd party/library methods as well as projects that still have to use Java 1.4.

The language selection is identical to the one described above. To select one or more parameters of a certain method, first choose its containing class either by typing in the name (the textfield supports completion) or by using the class chooser available through the [...] button. Next, select a method using the [...] button inside the "Method-Name" pane (it's not possible to edit the method name manually). Once a method has been chosen, the table in the *Parameters* pane is populated with the parameters of the selected method. Select the check box in the first column to have the selected language injected into arguments passed for this parameter. Note that only parameters of type String can be selected.

There's already an IntelliJ IDEA-core (Inspection Gadgets rather) inspection that checks the arguments of some well-known methods to be a valid regular expression. However, IntelliLang can provide more detailed error messages and all the features of the Regular Expression Support plugin.

The XPath language that is configured by default for some of the standard XPath-APIs is provided by the XPathView + XSLT-Support plugin, which is available here .

Importing Configuration Entries

To import the injection configuration from another IntelliJ IDEA installation use the *Import* button from the toolbar and select the file "IntelliLang.xml" from /options or from a JAR file that contains some exported (via File | Export Settings) IntelliJ IDEA settings. After selecting the file, a new dialog displays the entries contained in that file. Use the Delete-button to remove all entries you don't want to import. Note that this will *not* change the selected configuration file.

This selective import-feature makes it easy to share certain configurations in a team without losing any local entries like when importing settings via the core File | Import Settings action.

To prevent inconsistent data, the import is only possible if the existing configuration is unchanged or has been saved with the *Apply* button.

Advanced Settings

The advanced configuration tab allows you to specify different names for the base-annotations that should be used. This helps to avoid any dependencies on foreign code where this is not desired or possible. The custom annotations should just provide the same properties as the original ones, i.e. value for all of them and an optional (default = "") prefix and suffix for the @Language replacement.

Here it's also possible to configure the kind of runtime checks the plugin should generate for the @Pattern validation:

- No instrumentation: No checks will be inserted and doesn't touch any compiled class files
- Instrumentation using assert: Pattern-validation is controlled with the -ea JVM switch and throws AssertionError. This is the recommended way due to the potentially negative impact on the performance for methods that are invoked very often.
- Instrumentation using IllegalArgument- (for method parameters) and IllegalStateExceptions (for return values). This is the same the @NotNull instrumentation of IntelliJ IDEA does.

Contributed Configurations

Additional configurations can be downloaded and imported from here. This community effect can help to minimize the configuration efforts and make IntelliLang even easier to use.

Configuring Libraries and Dependencies in Grails Applications

You can configure module libraries and dependencies in Grails applications the same way as you would configure them in any other application.

Select the appropriate link to configure a module library or a dependency:

- Creating a module, project or global library and adding it to module dependencies
- Adding dependencies on modules

Java SE

IntelliJ IDEA provides features that simplify development with Java SE.

- Tutorial: Creating, Running and Packaging Your First Java Application
- Tutorial: Debugging Your First Java Application
- Tutorial: Migrating to Java 8

Tutorial: Creating, Running and Packaging Your First Java Application

This tutorial illustrates a Java SE application development workflow. The age-old Hello, World application is used as an example.

- Before you start
- Creating a project
- Exploring the project structure
- Creating a package and a class
- Writing code for the HelloWorld class
  - Using a live template for the main() method
  - Using code auto-completion
  - Using a live template for println()
- Building and running the application
- Remarks: building and running applications
- Packaging the application in a JAR
  - Creating an artifact configuration for the JAR
  - Building the JAR artifact
- Running the packaged application
  - Creating a JAR Application run configuration
  - Executing the run configuration

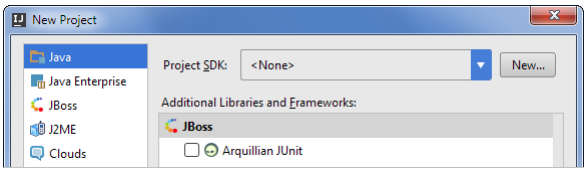
Before you start

To develop Java applications, you need a Java Development Kit (JDK). If you don't have a JDK installed, you can download an Oracle JDK from the Java SE Downloads page .

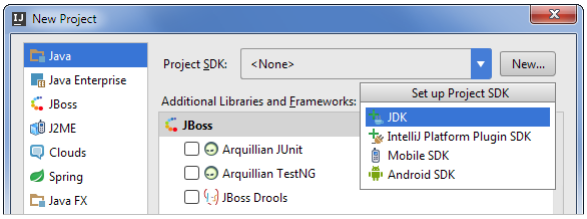
Creating a project

Any new development in IntelliJ IDEA starts with creating a project . So let's create one now.

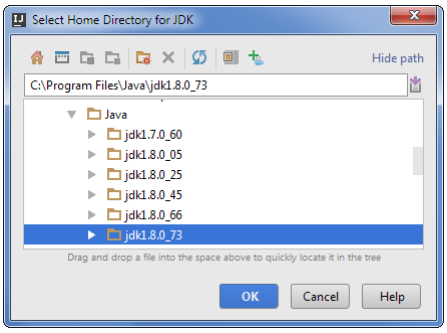
1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen. Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.
2. In the left-hand pane, select Java. (We want a Java-enabled project to be created, or, to be more exact, a project with a Java module .)



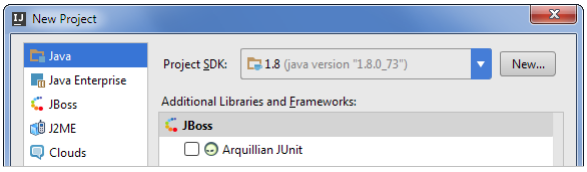
3. If you haven't defined a JDK in IntelliJ IDEA yet (in such a case <None> is shown the Project SDK field), you should do it now.
   Click New and select JDK.



In the Select Home Directory for JDK dialog that opens, select the directory in which you have the desired JDK installed.



The JDK you have specified is shown in the Project SDK field. This JDK will be associated by default with all projects and Java modules that you will create later.



Because our application is going to be a "plain old Java application", we don't need any "additional" technologies to be supported. So, don't select any of the options under Additional Libraries and Frameworks.

Click Next.

4. The options on the next page have to do with creating a Java class with a `main()` method.
   Since we are going to study the very basics of IntelliJ IDEA, and do everything "from scratch", we don't need these options at the moment. In other circumstances, they may, of course, be very useful.

   So, don't select any of the options.

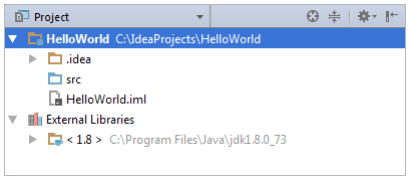
   Click Next.
5. On the next page, specify the project name (e.g. `HelloWorld`). If necessary, change the project location suggested by IntelliJ IDEA.

   Click Finish.

   Wait while IntelliJ IDEA is creating the project. When this process is complete, the structure of your new project is shown in the Project tool window.

Exploring the project structure

Let's take a quick look at the project structure.



There are two top-level nodes:

- HelloWorld. This node represents your Java module. The `.idea` folder and the file `HelloWorld.iml` are used to store configuration data for your project and module respectively. The folder `src` is for your source code.
- External Libraries. This is a category that represents all the "external" resources necessary for your development work. Currently in this category are the `.jar` files that make up your JDK.

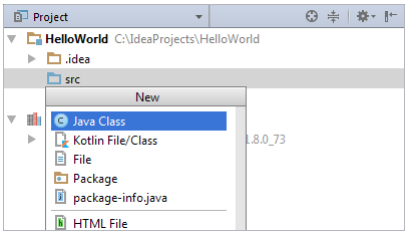
(For more information, see Tool Windows and Project Tool Window .)

Creating a package and a class

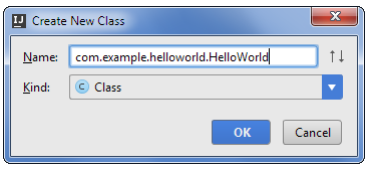
Now we are going to create a package and a class. Let the package and the class names be `com.example.helloworld` and `HelloWorld` respectively.

Though a package can be created separately, we will create both the package and the class at once.

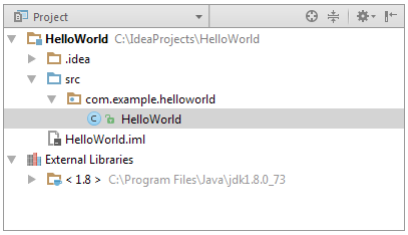
1. In the Project tool window, select the `src` folder and press `Alt+Insert`. (Alternatively, you can select File | New, or New from the context menu for the folder `src`.)
2. In the New menu, select Java Class (e.g. by pressing `Enter`).



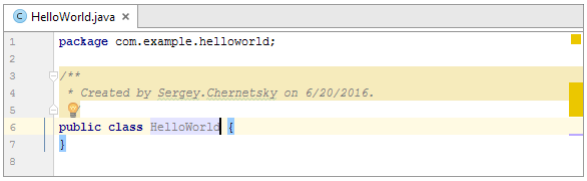
3. In the Create New Class dialog that opens, type `com.example.helloworld.HelloWorld` in the Name field. The Class option selected in the Kind list is OK for creating a class. Press `Enter` to create the package and the class, and close the dialog.



   The package `com.example.helloworld` and the class `HelloWorld` are shown in the Project tool window.

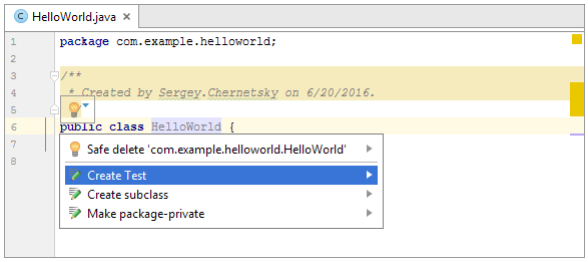


At the same time, the file `HelloWorld.java` (corresponding to the class) opens in the editor.



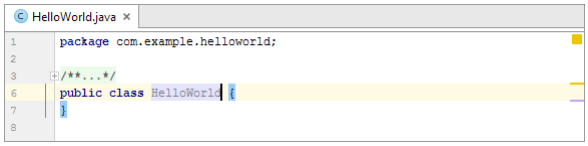
Note the package statement at the beginning of the file and also the class declaration. When creating the class, IntelliJ IDEA used a file template for a Java class. (IntelliJ IDEA provides a number of predefined file templates for creating various file types. For more information, see File and Code Templates .)

Also note a yellow light bulb. This bulb indicates that IntelliJ IDEA has suggestions for the current context. Click the light bulb, or press `Alt+Enter` to see the suggestion list.



At the moment, we are not going to perform any of the actions suggested by IntelliJ IDEA (such actions are called intention actions .) Note, however, that this IntelliJ IDEA feature may sometimes be very useful.

Finally, there are code folding markers next to the commented code block ( ). Click one of them to collapse the corresponding block. (For more information, see Code Folding .)



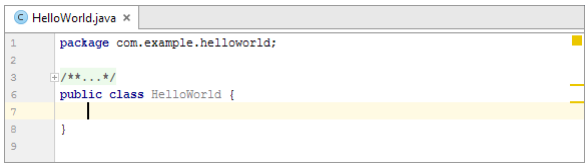
Writing code for the HelloWorld class

The code in its final state (as you probably know) will look this way:

```
package com.example.helloworld;

                    public class HelloWorld {
                    public static void main(String[] args)
                    {
                    System.out.println("Hello, World!");
                    }
                    }
```

The package statement and the class declaration are already there. Now we are going to add the missing couple of lines.

Press `Shift+Enter`. (In contrast to `Enter`, `Shift+Enter` starts a new line without breaking the current one.)

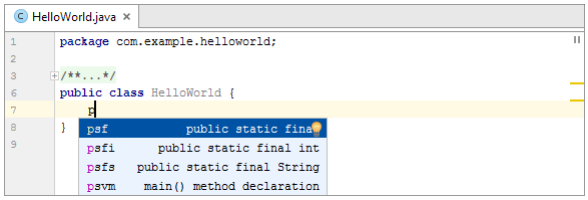


Using a live template for the main() method

The line

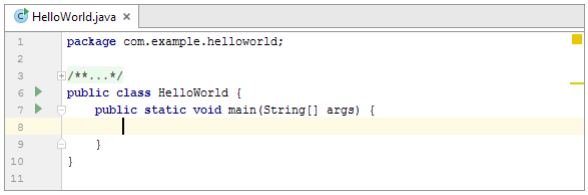
```
public static void
                    main(String[] args)
                    {}
```

may well be typed. However, we suggest that you use a different method. Type `p` and press `Ctrl+J`.



Select psvm - main() method declaration. (Use the `Up` and `Down` arrow keys for moving within the suggestion list, `Enter` for selecting a highlighted element.)

Here is the result:



(IntelliJ IDEA provides code snippets called live templates. `psvm` is an abbreviation for one of such templates. To insert a live template into your code, you can use Code | Insert Live Template or `Ctrl+J`. For more information, see Live Templates .)

Using code auto-completion

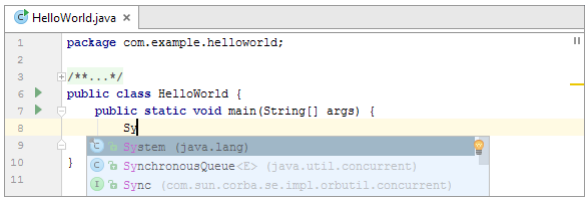
Now, it's time to add the remaining line of code

```
System.out.println("Hello, World!");
```

We'll do that using code auto-completion.

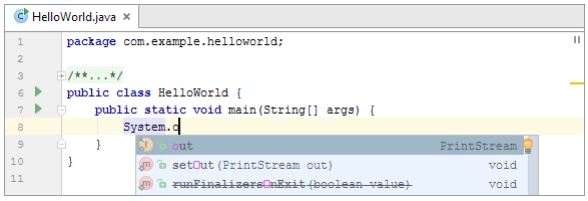
Type `Sy`

The code completion suggestion list is shown.



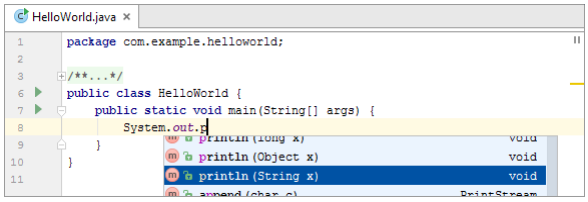
Select System (java.lang) by pressing `Enter`.

Type `.o` and press `Ctrl+Period`.

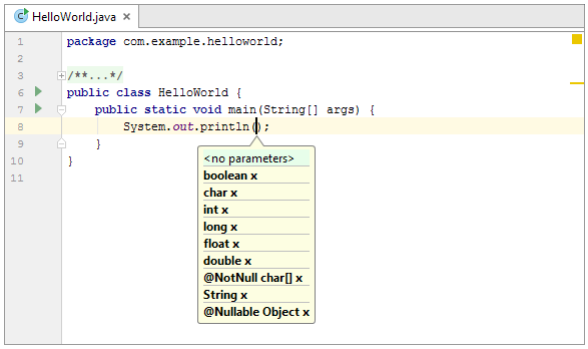


`out` is inserted followed by a dot. (You can select an item in the suggestion list by pressing `Ctrl+Period`. In that case, the selected item is inserted into the editor followed by a dot.)

Type `p` and then find and select println(String x).



IntelliJ IDEA prompts you which parameter types can be used in the current context.



Type `"`

The second quotation mark is inserted automatically and the cursor is placed between the quotation marks. Type `Hello, World!`

The code at this step is ready.

(For more information, see Auto-Completing Code .)

Using a live template for println()

As a side note, we could have inserted the call to `println()` by using a live template (`sout`).

If you think that it's enough for live templates, proceed to running the application . Otherwise, try that now as an additional exercise. Delete

```
System.out.println("Hello, World!");
```

Type `s` , press `Ctrl+J` and select sout - Prints a string to System.out.

The line

```
System.out.println();
```

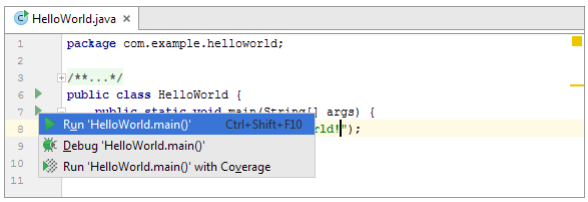
is added and the cursor is placed between `(` and `)` .

Type `"` and then type `Hello, World!`

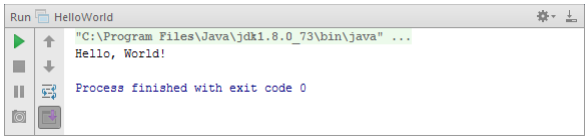
Building and running the application

Classes with a `main()` method can be run right from the editor. To show that, there are the green arrow markers ( ) in the left margin.

Click one of the markers and select Run 'HelloWorld.main()'.



Wait while IntelliJ IDEA is compiling the class. When the compilation is complete, the Run tool window opens at the bottom of the screen.



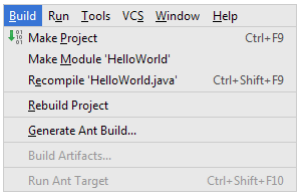
On the first line, there is a fragment of the command that IntelliJ IDEA used to run the class. (Click the fragment to see the whole command line including all options and arguments.) The last line shows that the process has exited normally, and no infinite loops occurred. And, finally, you see the program output `Hello, World!` between these lines.

(For more information, see Run Tool Window .)

Remarks: building and running applications

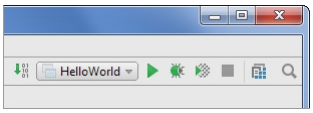
Some remarks related to building and running applications in IntelliJ IDEA:

– Prior to running the class, IntelliJ IDEA has automatically compiled it. When necessary, you can initiate the compilation yourself. The corresponding options can be found in the Build menu.



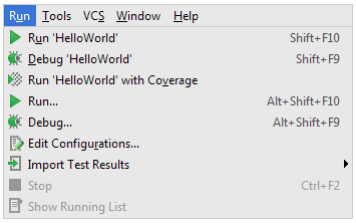
Many of these options are also available as context menu commands in the Project tool window and in the editor.

Finally, there is an icon in the upper-right part of the workspace that corresponds to the Make Project command ( ).



For more information, see Build Process and Compilation Types .

- The options for running and debugging applications can be found in the Run menu.



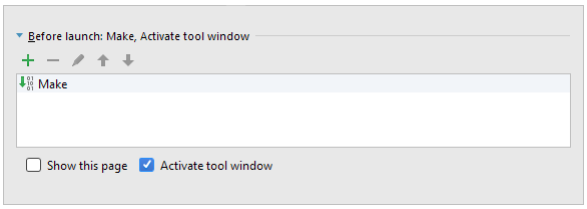
As in the case of the build operations, the run options can also be accessed from the Project tool window and the editor, as well as by means of controls in the upper-right part of the workspace.

- Applications in IntelliJ IDEA are run according to what is called run/debug configurations. Such configurations, generally, should be created prior to running an application.
  When you performed the Run 'HelloWorld.main()' command, IntelliJ IDEA, first, created a run configuration and then executed it.

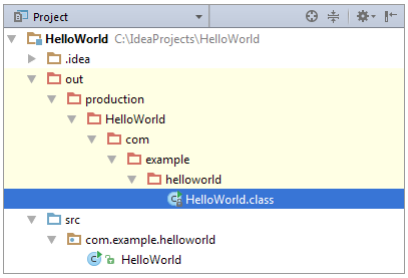
  The name of this run configuration (HelloWorld) is now shown in the run/debug configuration selector to the left of .

  The HelloWorld configuration now exists as a temporary configuration and, if necessary, you can save it to make it permanent.

- Run/debug configurations can do a lot more than just run applications. They can also build applications and perform other useful tasks.
  If you look at the settings for the HelloWorld run configuration (Run | Edit Configurations or Edit Configurations from the run configuration selector), you'll see that the Make option is included by default in the Before launch task list. That's why IntelliJ IDEA compiled the class when you performed the Run 'HelloWorld.main()' command.



- If you look at the Project tool window, you'll see that now there is the folder `out` there. This is the project output folder.
  Inside it is the module output folder (`production\HelloWorld`), the folder structure for the package `com.example.helloworld` and the compiled class file `HelloWorld.class` .



For more information, see Configuring Project Compiler Output and Configuring Module Compiler Output .

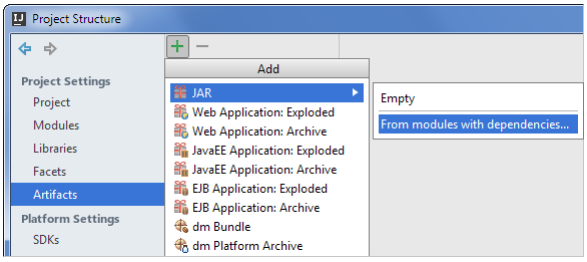
Packaging the application in a JAR

When happy with your application, you may want to package it in a Java archive (JAR) for distribution. To do that, you should create an artifact configuration for your JAR and then build the artifact.

Creating an artifact configuration for the JAR

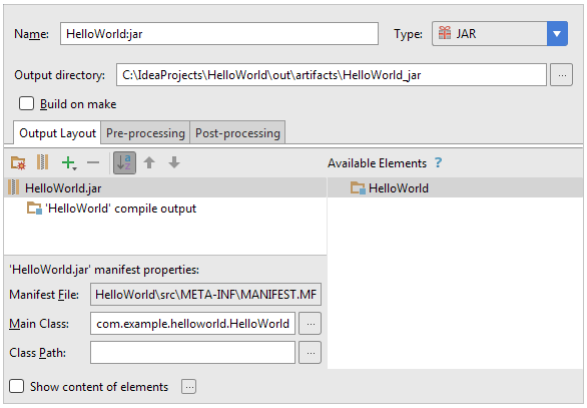
1. Select File | Project Structure to open the Project Structure dialog.
2. Under Project Settings, select Artifacts.

3. Click +, point to JAR and select From modules with dependencies.



4. In the dialog that opens, specify the main application class. (To the right of the Main Class field, click ... and select HelloWorld (com.example.helloworld) in the dialog that opens.)

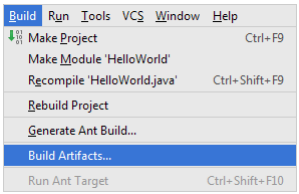
   As a result, the artifact configuration is created and its settings are shown in the right-hand part of the Project Structure dialog.



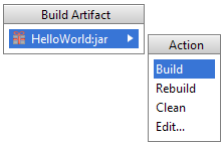
5. Click OK.

Building the JAR artifact

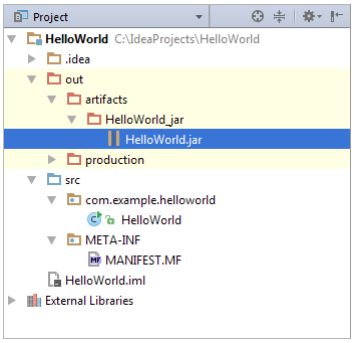
1. Select Build | Build Artifacts.



2. Point to HelloWorld:jar and select Build. (In this particular case, Build is the default action, so you can just press `Enter` instead.)



   If you now look at the `out/artifacts` folder, you'll find your JAR there.



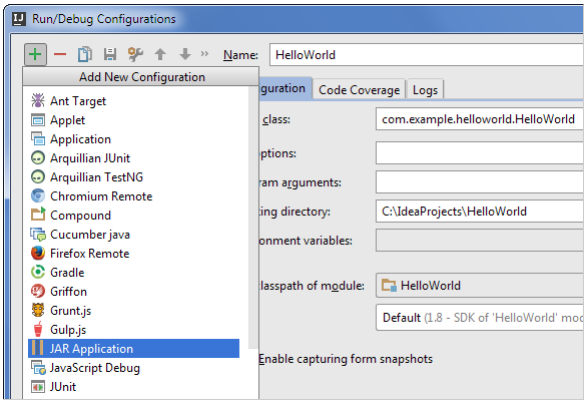
Running the packaged application

To make sure that everything is fine with the JAR, let's run it. To do that, we'll create a JAR Application run configuration and then execute that run configuration.

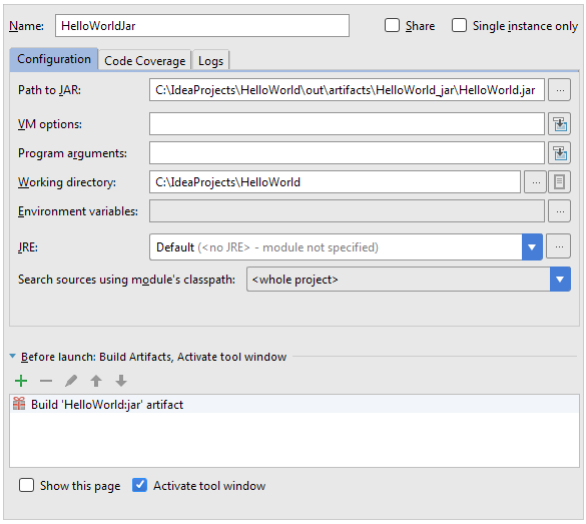
Creating a JAR Application run configuration

To run Java applications packaged in JARs, IntelliJ IDEA provides the JAR Application run configurations. To create such a configuration:

1. Select Run | Edit Configurations.
2. In the Run/Debug Configurations dialog that opens, click + and select JAR Application.

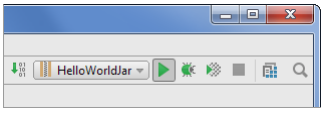


3. Specify the path to the JAR file. (To the right of the Path to JAR field, click ... and select the JAR file in the dialog that opens.)
   The rest of the settings in this case don't matter, however, there's one more thing that we'll do - just for convenience.
4. Under Before launch, click +, select Build Artifacts and select the HelloWorld:jar artifact in the dialog that opens.
   The Build 'HelloWorld:jar' artifact task is included in the Before launch task list. So each time you execute this run configuration, the artifact will be built automatically.

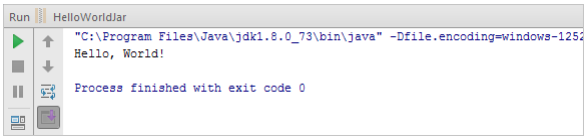


Executing the run configuration

− To the right of the run configuration selector, click ▶.



As before, the Run tool window opens and the application output is shown there.



Tutorial: Debugging Your First Java Application

On this page:

- Before you start...
- Putting breakpoints
- Starting a debugger session
- Stepping through the application
  - Stepping through the statements directly
  - Stepping through the method calls

Before you start...

You have already created and executed your first Java application . Now it's time to debug it.

However, it would be nice to add one more line to the application - let it be

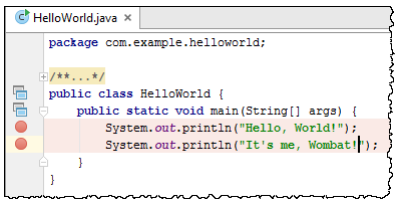
```
System.out.println("it's me, Wombat!");
```

Putting breakpoints

To start a debugger session, first of all you need to place a breakpoint at the statement where you want to suspend the execution of your application. The existing source code does not give you much of a choice - the only place, where you can put breakpoints, are the statements

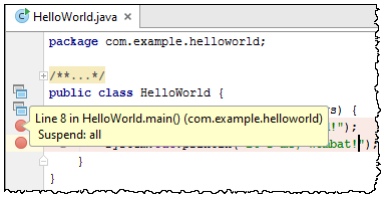
```
System.out.println("Hello World!");
                    System.out.println("it's me, Wombat!");
```

So let's do it. Click the left gutter at the line of the statement you want to put a breakpoint on, or just press `Ctrl+F8`:

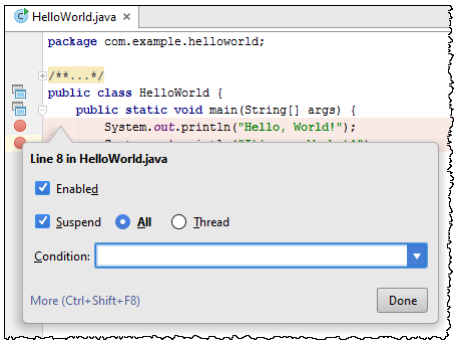


As you see, the new breakpoint is added to the source code. The line where the breakpoint is set changes its color to pink.

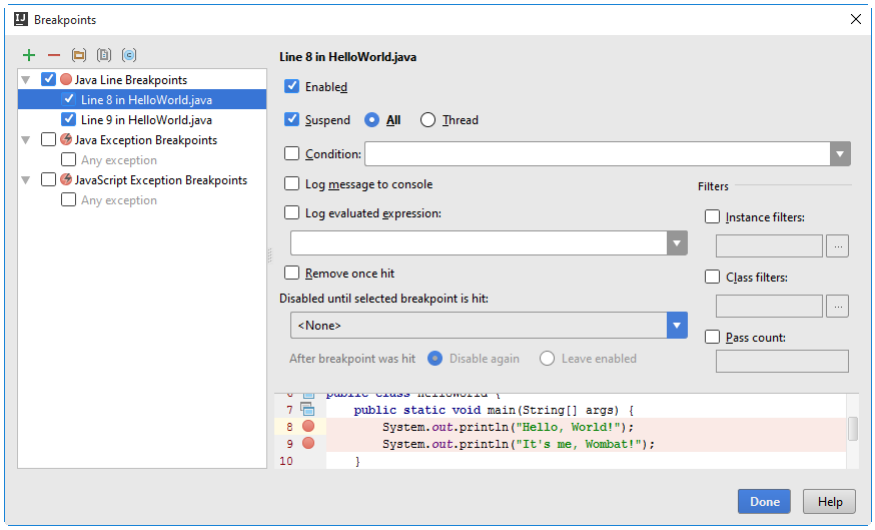
If you just hover your mouse pointer over the breakpoint, you will see its properties in the tooltip:



Suppose you want to change some properties of this breakpoint. Then right-click it and see the following dialog box:



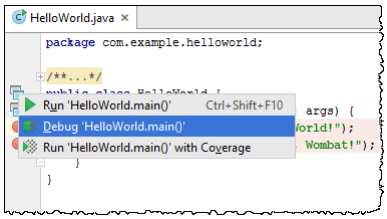
Finally, you would like to explore and change all the available properties of a breakpoint and see its location among the other breakpoints (if any). In this case, press `Ctrl+Shift+F8`:



Starting a debugger session

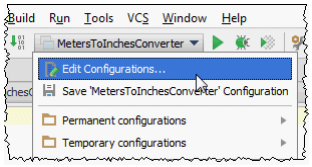
Now that the breakpoints are added, you can debug your application. This can be done in numerous ways; however, let's follow the easiest one.

Mind the icon that marks a class with the `main()` method. Clicking this icon reveals a menu that enables running and debugging such a class:

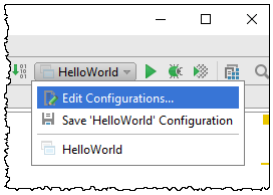


What does it mean?

IntelliJ IDEA launched the debugger session with the temporary run/debug configuration . This run/debug configuration has the default name "HelloWorld.main()'. To view and change the settings of this run/debug configuration, choose Run | Edit Configurations... on the main menu:



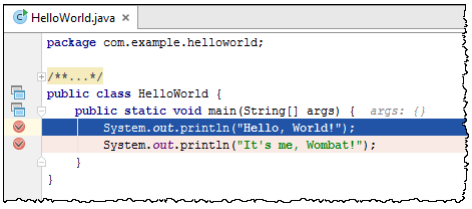
or click the run/debug configuration selector, and then choose Edit Configurations...:



IntelliJ IDEA compiles your application (which takes time!), and then suspends the application at the first breakpoint.

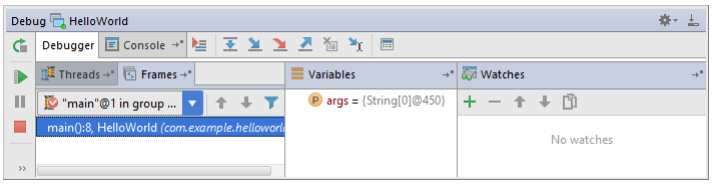
The IntelliJ IDEA window now looks different:

- The first thing that has changed is the color of the first line with the breakpoint. Now this line is rendered blue:



  It means that (according to the breakpoint properties) the application has reached this breakpoint, hit it and suspended before the statement `println` .

- Next, in the lower part of the IntelliJ IDEA window, a special tool window has emerged. This is the Debug tool window that features the stepping toolbar and shows all the necessary information you might need during the debugger session:



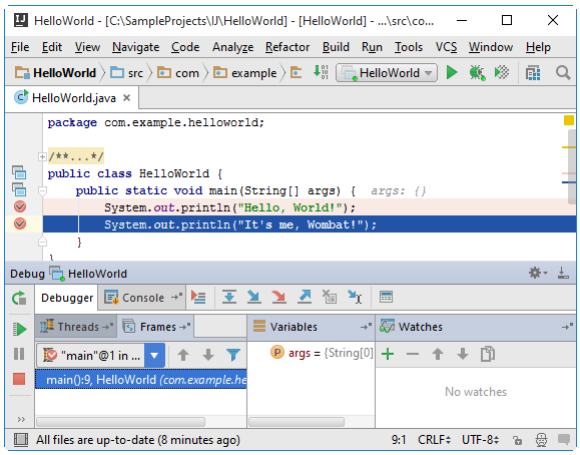
Stepping through the application

Stepping through the statements directly

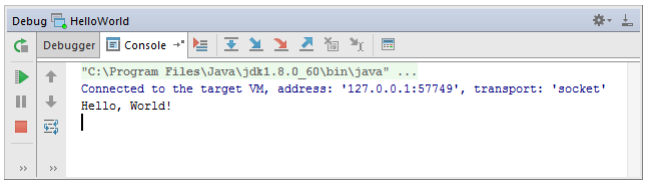
Let's step through the application. Click on the stepping toolbar, or just press F8.

The next line now becomes blue. If you look at the Debug tool window, you notice the following changes:

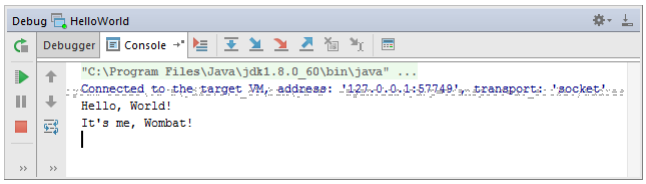
- In the Frames pane, the next line number is shown.
- The Console tab is marked with the icon, which means that it contains new output.



Click the Console tab. You see the message of the first line with the breakpoint "Hello, World!". The second message is not yet visible:



Click again on the stepping toolbar, or press F8. Now the second message appears in the console. After you click the next time, the application stops:



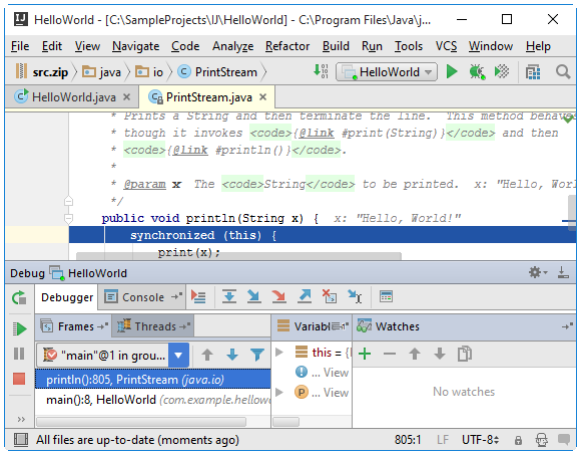
This debugging session is over.

Stepping through the method calls

Now let's explore a more complicated way and step into the `println()` call.

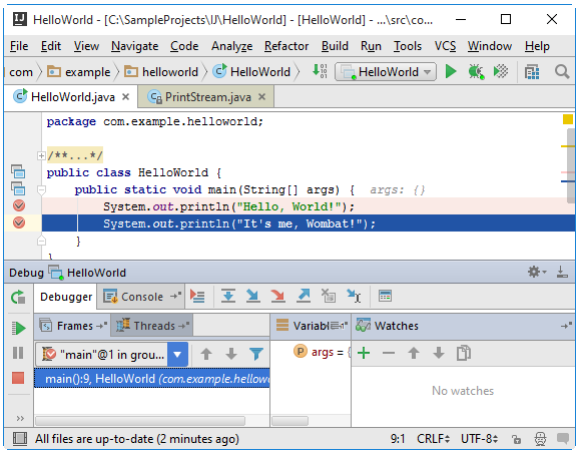
First, restart the debugger session. To do that, just click on the toolbar of the Debug tool window. Thus you'll rerun the latest run/debug configuration, namely, HelloWorld.

The application again pauses at the first breakpoint. This time, click , or press Shift+Alt+F7. You see a different picture:

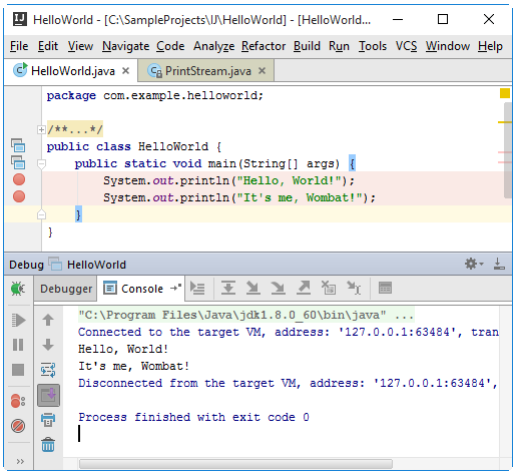


It means that IntelliJ IDEA has stepped into the method `println(String x)` of the library class `PrintStream.java`. Note that a new thread appears in the list of threads.

Click or press Shift+F8 to return to the next breakpoint:



Note that the Console tab has again got the marker, which means that new output is available. Next, click . You see that the process terminates:



Tutorial: Migrating to Java 8

Keeping your code up to date with the latest versions of languages and libraries is a challenging task. Fortunately, IntelliJ IDEA can make this easier, with inspections to guide your efforts, automatic fixes, and the usual refactoring tools.

Java SE 8 brings entire new concepts to the language, like lambda expressions, and adds new methods to classes that developers have been using comfortably for years. In addition, there are new ways of doing things, including the new Date and Time API, and an Optional type to help with null-safety.

In this tutorial we're going to show how IntelliJ IDEA can help you transition your code from Java 6 (or 7) to Java 8, using code examples to show what help is available and when you may, or may not, choose to use new features.

This tutorial assumes the following prerequisites:

- You already have an IntelliJ IDEA project for an existing codebase.

On this page:

- Approaching the problem
- Initial setup
- Configuring and running language level migration inspections
- Lambda expressions
- Impact of applying lambda expressions
- New Collection Methods
- Streams API - foreach
- Streams API - collect
- Impact of replacing foreach with Streams
- New Date and Time API
- Impact of migrating to the new Date and Time API
- Using Optional
- Impact of migrating to Optional
- Summary

Approaching the problem

The sheer number of options and features that IntelliJ IDEA has available might be overwhelming, especially when tackling a problem as big as trying to migrate a whole codebase (or even just a module or package) to a new version. As with most software development problems, it pays to approach this in an iterative fashion.

1. Pick a small number of changes to implement.
2. Pick a section of the codebase to apply them to.
3. Apply the changes in batches, running your project tests frequently and checking in to your VCS system when the tests are green.

To this end, this tutorial will group changes into sections rather than assume a Big Bang approach.

Initial setup

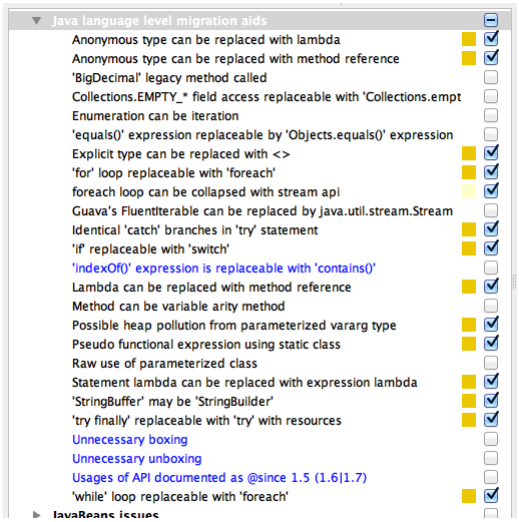
1. Make sure you're compiling with a Java 8 SDK. If you're not, change your SDK to the latest version of Java 8.
2. In the project settings , you should set your language level to "8.0 - Lambdas, type annotations".

If you are compiling the code in a CI environment, you'll need to ensure the new code is compiled using Java 8 there as well. Configuring this is beyond the scope of this tutorial.

Configuring and running language level migration inspections

Your project may already make use of inspections to encourage a certain level of consistency and quality in the code. To focus purely on just making changes related to upgrading to Java 8, we're going to create a new inspection profile.

1. Navigate to the inspections settings .
2. Create a new inspection profile called "Java8".
3. As a starting point for this profile, deselect everything using the "reset to empty" button .
4. We're going to select a set of language migration inspections to point out sections of the code we might want to update:



These inspections will show us areas in your code where you may be able to use the following Java 8 features:

- Lambda expressions
- Method references
- New Collection methods
- Streams API

5. Click OK to save these settings to the "Java8" profile and close the settings window.

6. Run the inspections , selecting the "Java8" profile and the scope to run the inspections on. If your project is small, that might be the whole codebase, but more likely you will want to select a module or package to start with.

Once Analyse code has finished running, you'll see a set of results in the Inspection Tool Window .

Lambda expressions

The inspections will show you places where you can convert code automatically to use lambda expressions. There are a number of places you might typically discover this in your existing code, for example when you create anonymous inner classes for:

- Runnable , Callable
- Comparator
- FileFilter , PathMatcher
- EventHandler
- Third party interfaces like Guava's Predicate

1. In the Inspection Tool Window , you should see results grouped under "Java language level migration aids". Under this heading, you may see "Anonymous type can be replaced with lambda". Open up this heading to see all the sections of the code where IntelliJ IDEA has detected you can use a lambda. You might see something like this:

2. For example, you may come across a `Runnable` anonymous inner class:

```
executorService.scheduleAtFixedRate(new Runnable() {
                    @Override
                    public void run() {
                    getDs().save(new
                    CappedPic(title));
                    }
                    }, 0, 500,
                    MILLISECONDS);
```

3. Many inspections suggest a fix that can be applied, and "Anonymous type can be replaced with lambda" does have a suggested resolution. To apply the fix, either:
   - Click on the Problem Resolution in the right of the inspection window, in our case this is Replace with lambda.
   - Or press `Alt+Enter` on the grey code in the editor and select Replace with lambda.

4. IntelliJ IDEA will then automatically change the code above to use a lambda expression:

```
executorService.scheduleAtFixedRate(()
                    -> getDs().save(new CappedPic(title)),
                    0, 500, MILLISECONDS);
```

You'll notice that lambda expressions can state very little in terms of type information. Here, the fact that this lambda represents an implementation of `Runnable` all but disappears. IntelliJ IDEA will provide you with information about the type of the lambda expression via the lambda icon in the left gutter:

Hovering over this will tell you the type, and clicking lets you navigate to the declaration.

Impact of applying lambda expressions

You should be able to automatically apply this fix to all places where anonymous inner classes are found in your codebase without impacting the functionality in your system. Applying the change will generally also improve the readability of your code, removing lines of boilerplate like in the example above.

However, you may want to check each individual change, as:

- Larger anonymous inner classes may not be very readable in a lambda form.
- There may be additional changes and improvements you can make.

Let's address both points with an example.

We might be using a `Runnable` to group a specific set of assertions in our test:

```
Runnable runnable =
                    new Runnable() {
                    @Override
                    public void run() {
                    datastoreProvider.register(database);
                    Assert.assertNull(database.find(User.class, "id", 1).get());
                    Assert.assertNull(database.find(User.class, "id", 3).get());

                    User foundUser = database.find(User.class, "id", 2).get();
                    Assert.assertNotNull(foundUser);
                    Assert.assertNotNull(database.find(User.class, "id", 4).get());
                    Assert.assertEquals("Should
                    find 1 friend", 1, foundUser.friends.size());
                    Assert.assertEquals("Should
                    find the right friend", 4, foundUser.friends.get(0).id);
                    }
                    };
```

Converting this to a lambda results in:

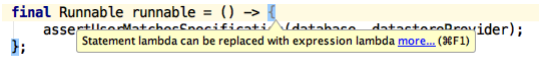
```
Runnable runnable =
                    () ->
                    {
                    datastoreProvider.register(database);
                    Assert.assertNull(database.find(User.class, "id", 1).get());
                    Assert.assertNull(database.find(User.class, "id", 3).get());

                    User foundUser = database.find(User.class, "id", 2).get();
                    Assert.assertNotNull(foundUser);
                    Assert.assertNotNull(database.find(User.class, "id", 4).get());
                    Assert.assertEquals("Should
                    find 1 friend", 1, foundUser.friends.size());
                    Assert.assertEquals("Should
                    find the right friend", 4, foundUser.friends.get(0).id);
                    };
```

This is not much shorter, nor does it impact readability much.

In cases like these, you may choose to use IntelliJ IDEA's extract method to pull these lines into a single method instead:

```
Runnable runnable =
                    () ->
                    {
                    assertUserMatchesSpecification(database,
                    datastoreProvider);
                    };
```

The second reason to check all your lambda conversions is that some lambdas can be further simplified. This last example is one of them - IntelliJ IDEA will show the curly braces in grey, and pressing `Alt+Enter` with the cursor on the braces will pop up the suggested change Statement lambda can be replaced with expression lambda:



Accepting this change will result in:

```
Runnable runnable =
                    () ->
                    assertUserMatchesSpecification(database,
                    datastoreProvider);
```

Once you've changed your anonymous inner classes to lambdas and made any manual adjustments you might want to make, like extracting methods or reformatting the code, run all your tests to make sure everything still works. If so, commit these changes to VCS. Once you've done this, you'll be ready to move to the next step.

New Collection Methods

Java 8 introduced a new way of working with collections of data, through the Streams API. What's less well known is that many of the `Collection` classes we're used to working with have new methods on them that are not via the Streams API. For example, `java.util.Iterable` has a `forEach` method that lets you pass in a lambda that represents an operation to run on every element. IntelliJ IDEA's inspections will highlight areas where you can use this and other new methods.

1. Back in the Inspection Tool Window , you should see "foreach can be collapsed with stream api" under "Java language level migration aids". You may not realise when you're going through all the inspections, but not all of these fixes will use the Streams API (more on Streams later). For example:

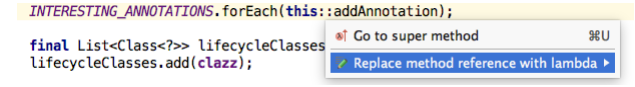
   ```
   for (Class<?
                       extends
                       Annotation > annotation : INTERESTING_ANNOTATIONS) {
                       addAnnotation(annotation);
                       }
   ```

   IntelliJ IDEA suggests "Can be replaced with foreach call". Applying this inspection gives us:

   ```
   INTERESTING_ANNOTATIONS.forEach(this::addAnnotation);
   ```

   Note that IntelliJ IDEA has applied all simplifications it could, going as far as using a Method Reference rather than a lambda. Method references are another new features in Java 8, which can generally be used where a lambda expression would usually call a single method.

2. Method references take a while to get used to, so you may prefer to expand this into a lambda to see the lambda version:

   Press `Alt+Enter` on the method reference and click Replace method reference with lambda. This is especially useful as you get used to all the new syntax. In lambda form, it looks like:

   ```
   INTERESTING_ANNOTATIONS.forEach((annotation) -> addAnnotation(annotation));
   ```

Both of the new forms do exactly the same thing as the original code - for every item in the `INTERESTING_ANNOTATIONS` list, it calls `addAnnotation` with the item.

Streams API - foreach

IntelliJ IDEA's inspections will suggest using the `forEach` on `Iterable` where appropriate, but it will also the new Streams API where this is a better choice.

The Streams API is a powerful tool for querying and manipulating data, and using it could significantly change and simplify the code you write. For this tutorial, we're going to look at some of the simplest use cases to get you started. Once you're more comfortable using this style of coding, you may then want to use its capabilities further.

1. What does the Streams API give us that we can't simply get from using a `forEach` method? Let's look at an example that's a slightly more complicated for loop than the previous one:

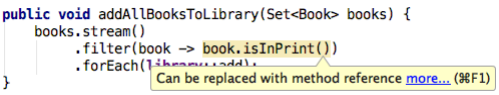
   ```
   public void addAllBooksToLibrary(Set<Book> books) {
                       for (Book book:
                       books) {
                       if (book.isInPrint()) {
                       library.add(book);
                       }
                       }
                       }
   ```

   Firstly the loop body checks some condition, then does something with the items that pass that condition.

2. Selecting the fix Replace with forEach will use the Streams API to do the same thing:

```
public void addAllBooksToLibrary(Set <Book> books) {
                    books.stream()
                    .filter(book ->
                    book.isInPrint())
                    .forEach(library::add);
                    }
```

In this case, IntelliJ IDEA has selected a method reference for the `forEach` parameter. For filter, IntelliJ IDEA has used a lambda, but will suggest in the editor that this particular example can use a method reference:



3. Applying this fix gives:

```
books.stream()
                    .filter(Book::isInPrint)
                    .forEach(library::add);
```

Streams API - collect

Instead of "can be replaced with foreach" call you might see "can be replaced with collect call". This is very similar to the above example, but instead of calling a `forEach` method at the end of the stream and performing some operation, this will use the stream's collect method to put all the results from the stream operation into a new `Collection`. It's very common to see a `for` loop that iterates over some collection, performs some sort of filtering or manipulating, and outputs the results into a new collection, and that's the sort of code this inspection will identify and migrate to using the Streams API.

1. In the Inspection Tool Window, you should see "foreach can be replaced with collect call" under "Java language level migration aids". Selecting one of these inspection results will show you a for loop that might look something like:

```
List <Key>
                    keys
                    = ....

                    List <Key.Id> objIds = new ArrayList<Key.Id>();
                    for (Key key :
                    keys) {
                    objIds.add(key.getId());
                    }
```

Here, we're looping over a list of Key objects, getting the Id from each of these objects, and putting them all into a separate collection of objIds.

2. Apply the Replace with collect fix to turn this code into:

```
List<Key.Id> objIds = keys.stream().map(Key::getId).collect(Collectors.toList());
```

3. Reformat this code so that you can see more clearly all the Stream operations:

```
List<Key.Id> objIds = keys.stream()
                    .map(Key::getId)
                    .collect(Collectors.toList());
```

This does exactly the same thing the original code did - takes a collection of `Key`s, "maps" each `Key` to its `Id`, and collects those into a new list, `objIds`.

Like the `forEach` example, IntelliJ IDEA can work out if a filter needs applying to a collect statement as well as maps, so it can cleverly turn many of your complex loops into a set of Stream operations.

Impact of replacing foreach with Streams

It may be tempting to run these inspections and simply apply all fixes automatically. When it comes to converting your code to use new methods on Collections or Streams, a little care should be taken. The IDE will ensure that your code works the same way it used to, but you need to check that your code remains readable and understandable after applying the changes. If you and your team are using Java 8 features for the first time, some of the new code will be very unfamiliar and probably unclear. Take the time to look at each change individually and check you're happy you understand the new code before going ahead.

Like with lambdas, a good rule of thumb is to start with small sections of code - short for loops that translate into two or fewer stream operations, preferably with single-line lambdas. As you become more familiar with the methods, then you may want to tackle more complex code.

Let's look at an example:

IntelliJ IDEA suggests that this code:

```
for (Entry<Class
                      <? extends Annotation>,
                      List<Annotation>> e : getAnnotations().entrySet()) {
                      if (e.getValue() != null && !e.getValue().isEmpty()) {
                      for (Annotation annotation: e.getValue())
                      {
                      destination.addAnnotation(e.getKey(), annotation);
                      }
                      }
                      }
```

Can be converted to this code:

```
getAnnotations().entrySet()
                      .stream()
                      .filter(e ->
                      e.getValue() !=
                      null && !e.getValue().isEmpty())
                      .forEach(e ->
                      {
                      for (Annotation annotation: e.getValue())
                      {
                      destination.addAnnotation(e.getKey(), annotation);
                      }
                      });
```

Setting aside the fact that the original code is challenging to understand to begin with, you may choose not to apply the changes for a number of reasons:

- Despite refactoring away the outer-loop, there's still a for loop inside the `forEach` method. This suggests that there may be a different way to structure the stream call, perhaps using flatMap .
- The `destination.addAnnotation` method suggests that there may be a way to restructure this to use a `collect` call rather than a `forEach` .
- It's arguably not easier to understand than the original code.

However, you may choose to accept this change for the following reasons:

- This is a complex piece of code that is iterating through and manipulating data in a collection, therefore a move towards the Streams API is a move in the right direction. It can be further refactored or improved later when the team's developers are more familiar with the way Streams work.
- In the new code the `if` condition has been moved into a `filter` call, making clearer what purpose this section of the code is.

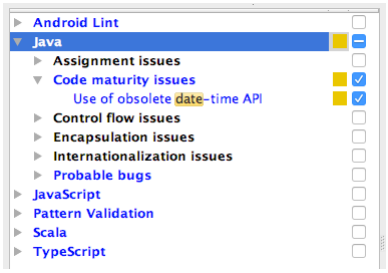
Apart from the options "keep the code" and "apply the changes", there's a third option: refactor the old code to something more readable, even if it doesn't use Java 8. This might be a good piece of code to make a note of to refactor later, rather than trying to tackle all the code's problems while simply trying to adopt more Java 8 conventions.

New Date and Time API

The inspections we've selected for our "Java8" profile help us to locate places where we can use lambda expressions, new methods on Collections and the Streams API, and will apply fixes automatically to those places. There are plenty of other new features in Java 8, and in the following sections we'll highlight some features of IntelliJ IDEA that may help you use these too.

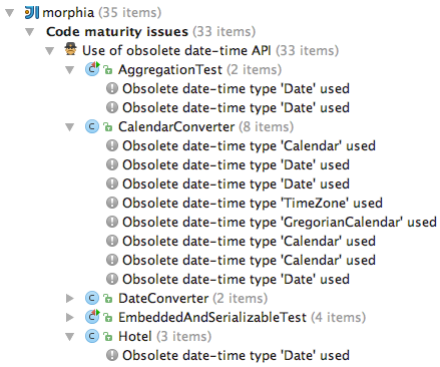
In this section, we'll look at locating places that may benefit from using the new Date and Time API instead of `java.util.Date` and `java.util.Calendar` .

1. You'll need to enable a new inspection to locate uses of the old Date and Time API.



   Note that although many methods have been deprecated on `java.util.Date` for some time, the class itself is not deprecated, so if you use it in your code you will not receive deprecation warnings. That's why this inspection is useful to locate usages.

2. Run the inspection . You should see a list of results that looks something like this:



3. Unlike the earlier inspections, these do not have suggested fixes as they will require you and your team to evaluate the use of the old classes and decide how to migrate them to the new API. If you have a `Date` field that represents a single date without a time, for example:

```
public class HotelBooking
                    {
                    private final Hotel hotel;
                    private final Date checkInDate;
                    private final Date checkOutDate;

                    // constructor, getters and setters...
                    }
```

   you may choose to replace this with a `LocalDate`. This can be done via the context menu Refactor | Type Migration... or via `Ctrl+Shift+F6`. Type LocalDate in the popup and select `java.time.LocalDate`. When you press enter, this will change the type of this field and getters and setters. You may still need to address compilation errors where the field, getters or setters are used.

4. For fields that are both date and time, you may choose to migrate these to `java.time.LocalDateTime`. For fields that are only time, `java.time.LocalTime` may be appropriate.

5. If you were setting the original values with a new `Date`, knowing that this is the equivalent to the date and time right now:

```
booking.setCheckInDate(new Date());
```

   you can instead use the `now()` method:

```
booking.setCheckInDate(LocalDate.now());
```

6. A common and readable way to set a value for `java.util.Date` was to use `java.text.SimpleDateFormat`. You might see code that looks something like:

```
SimpleDateFormat format =
                    new
                    SimpleDateFormat("yyyy-MM-dd");
                    booking.setCheckInDate(format.parse("2017-03-02"));
```

   If this check in date has been migrated to a `LocalDate`, you can easily set this to the specific date without the use of a formatter:

```
booking.setCheckInDate(LocalDate.of(2017, 3, 2));
```

These examples barely scratch the surface of the changes you may want or need to do in order to fully utilise the new date and time features in Java 8. Take a look at the tutorial provided by Oracle for more information on the new API features and how to use them.

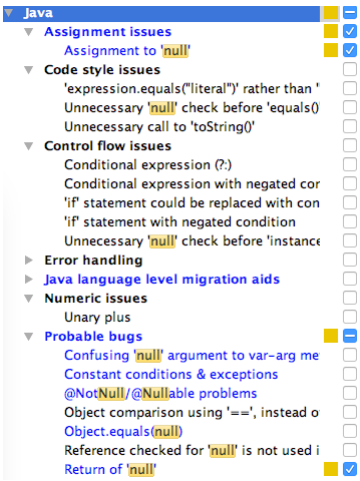
Impact of migrating to the new Date and Time API

Updating your code to use the new Date and Time API requires much more manual intervention than migrating anonymous inner classes to Lambda Expressions and loops to the Streams API. IntelliJ IDEA will help you see how much and where you use the old `java.util.Date` and `java.util.Calendar` classes, which will help you understand the scope of the migration. IntelliJ IDEA's refactoring tools can help you migrate these types if necessary. However, you will need to have a strategy on how to approach each of the changes, which new types you want to use, and how to use these correctly. This is not a change you can apply automatically.

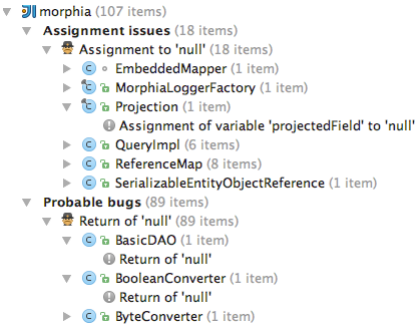
Using Optional

The last Java 8 feature we'll look at is the new Optional type. `java.util.Optional` gives you a way to handle null values, and a way to specify if a method call is expected to return a null value or not. Like Date and Time, IntelliJ IDEA's features will help you to identify areas of your code that might benefit from using the `Optional` type.

1. There are a number of inspections that look for the use nulls in Java code, these can be useful for identifying areas that may benefit from using `Optional`. We'll look at enabling just two of these inspections for simplicity:



2. Run the code analysis . You should see a list of results that looks something like this:



3. If you see "Assignment to null" for fields, you may want to consider turning this field into an `Optional` . For example, in the code below, the line where offset is assigned will be flagged:

```
private Integer offset;

                    // code....

                    public Builder offset(int value) {
                    offset = value > 0
                    ? value : null;
                    return this;
                    }

                    // more code...
```

That's because in another method, the code checks to see if this value has been set before doing something with it:

```
if (offset !=
                    null) {
                    cursor.skip(offset);
                    }
```

In this case, null is a valid value for offset - it indicates this has not been set, and therefore shouldn't be used. You may wish to change the field into an `Optional` of `Integer` via `Ctrl+Shift+F6` , and alter the way the value is set:

```
private Optional<Integer> offset;

                    // code...

                    public Builder offset(int value) {
                    offset = value > 0
                    ? Optional.of(value) : Optional.empty();
                    return this;
                    }

                    // more code...
```

Then you can use the methods on `Optional` instead of performing null-checks. The simplest solution is:

```
if (offset.isPresent()) {
                    cursor.skip(offset);
                    }
```

But it's much more elegant to use a Lambda Expression to define what to do with the value:

```
offset.ifPresent(() ->
                    cursor.skip(offset));
```

4. The inspections also indicate places where a method returns null. If you have a method that can return a null value, the code that calls this method should check if it returned null and take appropriate action. It's easy to forget to do this though, especially if the developer isn't aware the method can return a null. Changing these methods to return an `Optional` makes it much more explicit this might not return a value. For example, maybe our inspections flagged this method as returning a null value:

```
public Customer findFirst()
                    {
                    if (customers.isEmpty()) {
                    return null;
                    } else {
                    return customers.get(0);
                    }
                    }
```

   We could alter this method to return an `Optional` of `Customer`:

```
public Optional<Customer> findFirst() {
                    if (customers.isEmpty()) {
                    return Optional.empty();
                    } else {
                    return Optional.ofNullable(customers.get(0));
                    }
                    }
```

5. You'll need to change the code that calls these methods to deal with the `Optional` type. This might be the correct place to make a decision about what to do if the value does not exist. In the example above, perhaps the code that calls the `findFirst` method used to look like this:

```
Customer firstCustomer =
                    customerDao.findFirst();
                    if (firstCustomer == null) {
                    throw new CustomerNotFoundException();
                    } else {
                    firstCustomer.setNewOffer(offer);
                    }
```

   But we're now returning an `Optional`, we can eliminate the null check:

```
Optional<Customer> firstCustomer = customerDao.findFirst();
                    firstCustomer.orElseThrow(()
                    -> new CustomerNotFoundException())
                    .setNewOffer(offer);
```

Impact of migrating to Optional

Changing a field type to `Optional` can have a big impact, and it's not easy to do everything automatically. To start with, try to keep the use of `Optional` inside the class - if you can change the field to an `Optional` try not expose this via getters and setters, this will let you do a more gradual migration.

Changing method return types to `Optional` has an even bigger impact, and you may see these changes ripple through your codebase in an unexpected way. Applying this approach to all values that can be null could result in `Optional` variables and fields all over the code, with multiple places to performing `isPresent` checks or using the `Optional` methods to perform an action or throw an appropriate exception.

Remember that the goal of using the new features in Java 8 is to simplify the code and aid readability, so limit the scope of the changes to small sections of the code and check that using `Optional` is making your code easier to understand, not more difficult to maintain.

IntelliJ IDEA's inspections will identify possible places for change, and the refactoring tools can help apply these changes, but refactoring to `Optional` has a large impact and you and your team should identify a strategy for which areas to change and how to approach these changes. You can even use the suggested fix of "Annotate field [fieldName] as @Nullable" to mark those fields that are candidates for migrating to `Optional`, in order to take a step in that direction with a smaller impact on the code.

Summary

IntelliJ IDEA's Inspections, in particular those around language migration, can help identify areas in your code that can be refactored to use Java 8 features, and even apply those fixes automatically.

If you have applied the fixes automatically, it's valuable to look at the updated code to check it isn't harder to understand, and to help you become familiar with the new features.

This tutorial gave some pointers on how to migrate your code. We've covered lambda expressions and method references, some new methods on Collection, introduced the Streams API, shown how IntelliJ IDEA can help you use the new Date and Time API and looked at how to identify places that might benefit from using the new Optional type.

There are plenty of new features in Java 8 designed to make life easier for programmers - to make code more readable, and to make it easier to perform complex operations on data structures. IntelliJ IDEA of course not only supports these features, but helps developers make use of them, including migrating existing code and providing help and suggestions in the editor to guide you as you use them.

Java EE

IntelliJ IDEA provides facilities for developing applications using Java EE technologies.

- Enabling Java EE Application Support
- Working with Application Servers
- Working with Cloud Platforms
- Tutorial: Developing a Java EE Application

Enabling Java EE Application Support

Java EE application development support in a module is represented by a Java EE Application facet .

A module can have more than one Java EE Application facet.

- Adding a Java EE Application facet to a module
- Managing Java EE Application facet resources
- Fixing project configuration problems

Adding a Java EE Application facet to a module

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the leftmost pane, select Modules.
3. In the pane to the right, select the module of interest.
4. Click + (`Alt+Insert`) and select JavaEE Application.
5. Click OK in the Project Structure dialog.

Managing Java EE Application facet resources

Java EE application facet resources are initially represented by the deployment descriptor `application.xml` file. You can replace this file with a different one or regenerate this file. You can also create application-server-specific deployment descriptors if necessary.

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the leftmost pane, select Modules.
3. In the pane to the right, expand the module of interest and select its Java EE application facet (shown as javaEEApplication).
4. On the Java EE Application Facet page that open in the right-hand part of the dialog:
   - To replace the deployment descriptor file with a different `application.xml` file: under Deployment Descriptors, select the file and click . In the dialog that opens, specify the path to the necessary file and click OK.
   - To regenerate `application.xml` : delete the existing file (–) and then create a new file (+).
   - To create an application-server-specific descriptor file: click Add Application Server specific descriptor. In the dialog that opens, select the server and the associated options, and click OK.
     Once the file has been created, you can perform with it the same operations as with `application.xml` .

Fixing project configuration problems

Sometimes there may be project configuration problems related to the use of the Java EE application facet. IntelliJ IDEA informs you about such problems on the Java EE Application Facet page of the Project Structure dialog and suggests fixes. For example:

- If the facet resources are not included in any of the existing project artifacts , click the Fix button and select the corresponding artifact from the list.
- If no appropriate artifacts are currently configured, click the Create Artifact button and configure the artifact layout on the Artifacts page of the Project Structure dialog that opens.
- If missing libraries are detected, click the Fix button to download the corresponding libraries.

Object-Relational Mapping (EJB, Hibernate and JPA)

IntelliJ IDEA enables object/relational mappings by means of the dedicated facets:

- Hibernate
- JPA
- EJB

For the details of EJB support, refer to the section EJB . Support of Hibernate and JPA technologies in IntelliJ IDEA includes:

- Automatic creation of the specific module structure and the necessary descriptors.
- Possibility to automatically download missing libraries.
- Support of Java beans .
- Mapping data sources , configured in IntelliJ IDEA, to the Hibernate session factories, JPA persistence units, or entity beans.
- Coding assistance, including code completion, syntax and error highlighting. IntelliJ IDEA recognizes specific syntax, highlights invalid references and other errors, and suggests quick fixes.
- Refactoring source code.
- Using Hibernate and JPA annotations.
- Possibility to view relationships in the Entity Relationship diagram .
- Templates to support configuration files.
- Surround with () for HQL/JPAQL (`Ctrl+Alt+T`)

Note Hibernate and JPA facets are often used in combination with the other facets, for example Spring or Seam .

Working with Application Servers

IntelliJ IDEA can be integrated with the "most popular" application server systems. You can deploy your application artifacts onto the corresponding servers and debug the deployed applications right from within the IDE. You can also start and stop the servers installed on your computer.

- Supported Application Servers
- Main Tasks Related to Working with Application Servers
- Enabling Application Server Integration Plugins
- Defining Application Servers in IntelliJ IDEA
- Working with Server Run/Debug Configurations
- Updating Applications on Application Servers

Supported application servers

IntelliJ IDEA provides integration with the following application servers:

- CloudBees
- Geronimo
- GlassFish
- Google App Engine Development Server
- JBoss
- Jetty
- JSR45-compatible application servers (any server that supports JSR-45)
- Resin
- SpringSource dm Server
- Tomcat
- TomEE
- WebLogic
- WebSphere (starting from version 6.1)

Main tasks related to working with application servers

Here are main tasks related to working with application servers in IntelliJ IDEA.

1. Download and install the server that you are going to use. Even if you are going to use a remote server (i.e. one running on a different computer), the same server version must be installed locally.
2. Make sure that the necessary server integration plugin or plugins are enabled in IntelliJ IDEA. See Enabling Application Server Integration Plugins .
3. Define the server in IntelliJ IDEA. See Defining Application Servers in IntelliJ IDEA .
4. Create an artifact configuration for your application.
5. Create a run/debug configuration for your server. See Creating a server run/debug configuration.
6. Start the server run/debug configuration to run or debug your application. See Starting a server run/debug configuration.

Enabling application server integration plugins

All the available server integration plugins are enabled by default. However, before you start working with an application server, it's always worth making sure that the necessary plugin or plugins are enabled.

The server integration plugin names, normally, include the name of the server and the word Integration or Support, for example, GlassFish Integration, dmServer Support, Tomcat and TomEE Integration, etc.

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, select Plugins.
3. In the right-hand part of the dialog, on the Plugins page , type the server name of interest (e.g. `tomcat`) in the search box. As a result, only the plugins whose names and descriptions contain the typed text are shown in the list of plugins.
4. If the check box to the right of the plugin name is not selected, select it.
5. Click OK in the Settings dialog.
6. If suggested, restart IntelliJ IDEA.

Defining Application Servers in IntelliJ IDEA

To define a server in IntelliJ IDEA, in most of the cases, all you have to do is to specify where the corresponding server is installed.

You can define a server:

- Separately, in the Settings dialog: `Ctrl+Alt+S` | Build, Execution, Deployment | Application Servers | +, etc.
- When creating a project or module .
- When creating a server run/debug configuration : Run | Edit Configurations | + | <Server Name>, etc.

On this page:

- Defining a server in the Settings dialog
- Defining a server when creating a project or module
- Defining a server when creating a run/debug configuration

Defining a server in the Settings dialog

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, in the Build, Execution, Deployment category, select Application Servers.
3. On the Application Servers page that opens in the right-hand part of the dialog, click +. (Alternatively, press `Alt+Insert` .)
4. Select the server that you are going to use.
5. In the dialog that opens, specify the server settings and click OK. For most of the servers, you have to specify just the server home, i.e. the server installation directory. For more information, see Application Servers .
6. Click OK in the Settings dialog.

Defining a server when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java Enterprise.
3. In the right-hand part of the page, to the right of the Application Server field, click New.
4. Select the server that you are going to use.
5. In the dialog that opens, specify the server settings and click OK. For most of the servers, you have to specify just the server home, i.e. the server installation directory.
6. Specify other settings as necessary and click Next. For more information, see Project Category and Options or Module Category and Options .
7. Specify the name and location settings and click Finish. For more information, see Project Name and Location or Module Name and Location .

Defining a server when creating a run/debug configuration

1. Open the Run/Debug Configurations dialog (e.g. Run | Edit Configurations).
2. Click + (`Alt+Insert`), select the server of interest (e.g. Tomcat Server) and, if available, select Local or Remote.
3. In the right-hand part of the dialog, on the Server tab, click Configure to the right of the Application server list.
4. In the dialog that opens specify the server settings and click OK.
5. Specify other run/debug configuration settings as necessary and click OK.

Working with Server Run/Debug Configurations

To run or debug your Java EE or Web application on an application server, you need an application server run/debug configuration .

One such configuration may be created by IntelliJ IDEA automatically. This happens if you, when creating a project or module, specify an application server that you are going to use (see Defining a server when creating a project or module).

You can create more server run/debug configurations if and when needed.

- Local and remote run configurations
- What happens when a server run configuration is started
- Creating a server run/debug configuration
- Starting a server run/debug configuration

Local and remote run configurations

A server run/debug configuration may be local or remote.

Local configurations are for servers installed on your computer. Such configurations include the settings that define how the corresponding server is to be started. Consequently, when you execute a local configuration, IntelliJ IDEA, among other things, starts the server.

Remote configurations don't start a server. Usually, they are used for servers running on different (remote) computers. They may as well be used for servers installed locally (on your computer) in cases when you don't want the run configuration to start (or stop) the server.

When you execute a remote configuration, IntelliJ IDEA connects to the server to be able to deploy application artifacts or to perform their debugging. At that time, the corresponding server must already be running.

To conclude, what principally distinguishes local and remote configurations is not where the server is physically installed (though this is also important) but whether or not the server is started (or stopped) by means of the corresponding run configuration.

What happens when a server run configuration is started

When you start a server run/debug configuration, IntelliJ IDEA, usually, does the following:

1. Performs the Before launch tasks. By default, these are Make and Build Artifacts: IntelliJ IDEA compiles the project and builds the application artifacts. The Build Artifacts task is not initially present in a run configuration but added automatically as soon as you specify the artifacts to be deployed onto the server.
2. Starts the server (for a local configuration) or connects to the server (for a remote configuration). (At that moment, the Run or the Debug tool window opens so that you can monitor and control the process.)
3. Deploys the specified artifacts (and/or deployable components external to your project) to the server.
4. If so specified, starts a web browser and opens a specified URL (which usually corresponds to a starting page of your application).

Creating a server run/debug configuration

1. Open the Run/Debug Configurations dialog (e.g. Run | Edit Configurations).
2. Click + (`Alt+Insert`), select the server of interest (e.g. Tomcat Server) and, if available, select Local or Remote. (See Local and remote run configurations.)
3. In the right-hand part of the dialog, specify the run/debug configuration settings and click OK. (For information on the available settings, see the corresponding server-specific topic in Run/Debug Configurations).

See also, Creating and Editing Run/Debug Configurations .

Starting a server run/debug configuration

An application server run/debug configuration can be started in a usual way, as any other run configuration. See Running Applications and Starting the Debugger Session .

Alternatively, you can use the Application Servers Tool Window :

1. Open the Application Servers tool window (e.g. View | Tool Windows | Application Servers).
2. Select the server run/debug configuration that you want to use.
3. On the toolbar of the tool window, click:
   - ▶ to start the selected configuration in the run mode.
   - 🐞 to start the selected configuration in the debug mode.

   As a result, the Run or the Debug tool window opens.

Updating Applications on Application Servers

When running or debugging a Java EE or Web application, you can modify the source code and, almost immediately, see the result of your changes.

- Updating an application: Process overview
- Specifying application update options
- Updating an application
- Application update options

Updating an application: Process overview

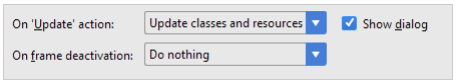
1. Specify the necessary application update options in your server run/debug configuration, see Specifying application update options.
2. Start the run/debug configuration.
3. After making changes to the source code, update your application, see Updating an application.

Specifying application update options

In server run/debug configurations, the following settings on the Server tab have to do with updating an application:

- On 'Update' action. The *Update action* refers to clicking in the Run or the Debug tool window (alternatively, `Ctrl+F10` or Run | Update '<app name>' application). Use the list to select the default update option. See Application update options.

  Show dialog. If this check box is not selected, the default update option is used. Otherwise, a dialog is shown that displays all the available update options, and you'll be able to select the necessary option prior to actually updating your application.



- On frame deactivation. *Frame deactivation* means switching from IntelliJ IDEA to a different application (e.g. a Web browser). Use the list to specify what IntelliJ IDEA should do in such cases.

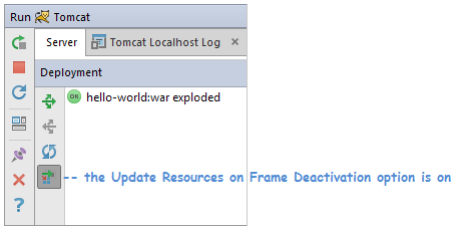
Updating an application

When the Run or the Debug tool window is active, do one of the following:

- Click on the toolbar of the tool window.
- Press `Ctrl+F10`.
- Select Run | Update '<app name>' application.

If the necessary update option is associated with frame deactivation , your application is updated automatically when you switch from IntelliJ IDEA to a different application (e.g. a Web browser).

Note that you can turn the Update Resources on Frame Deactivation option on and off right in the Run or the Debug tool window. To do that, click.



You can also update your application in the Application Servers Tool Window by redeploying your application artifact ( ).

Application update options

The update options are different depending on:

- the artifact format, i.e. on whether the application artifact is exploded (unpacked) or packed (e.g. WAR, EAR)
- the run/debug configuration type, i.e. on whether the run/debug configuration is local or remote (see Local and remote run configurations)

Option	Description	Available for
Update resources	All changed resources are updated (HTML, JSP, JavaScript, CSS and image files).	Exploded artifacts in local configurations
Update classes and resources	Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used.	Exploded artifacts in local configurations
Hot swap classes	Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode.	Packed artifacts in local configurations; exploded and packed artifacts in remote configurations
Redeploy	The application artifact is rebuilt and redeployed. The operation may be time-consuming.	Exploded and packed artifacts in local and remote configurations
Restart server	The server is restarted. The application artifact is rebuilt and redeployed. The operation may be very time-consuming.	Exploded and packed artifacts in local configurations

Working with Cloud Platforms

- Supported cloud platforms
- Overview of the cloud support
- Working with a cloud platform: Process overview
- Cloud integration plugins
- Cloud run configurations

Supported cloud platforms

IntelliJ IDEA provides integration with the following cloud platforms:

- CloudBees
- Cloud Foundry
- Google App Engine
- Heroku
- OpenShift

Overview of the cloud support

IntelliJ IDEA lets you:

- Deploy your application artifacts to the clouds.
- Deploy your code to Heroku and OpenShift.
- Debug applications on Heroku and OpenShift.

For all such purposes, IntelliJ IDEA provides corresponding cloud run/debug configurations .

You can also monitor and control the deployment process, and view your application logs in the Application Servers Tool Window .

Working with a cloud platform: Process overview

1. Sign up for a user account at the cloud service provider website.
2. Make sure that the corresponding cloud integration plugin is enabled.
3. Register your cloud user account in IntelliJ IDEA. You can do that:
   - Separately, in the Settings dialog: Ctrl+Alt+S | Build, Execution, Deployment | Clouds | +, etc.
   - When creating a project or module for working with a cloud, e.g. File | New | Project | Clouds, etc.
   - When creating a cloud run configuration: Run | Edit Configurations | + | <Cloud Name> Deployment, etc.

   For information on cloud user account settings, see Clouds .
4. Create an artifact configuration for your application. (For Heroku or OpenShift, this may be unnecessary. You can deploy your source code to the cloud, and the cloud platform will do the rest of the work for you. For CloudBees and Cloud Foundry, IntelliJ IDEA can create the necessary configurations automatically.)
5. Build the artifact. You can do that separately (Build | Build Artifacts), or when executing the corresponding run configuration. (In the case of deploying code to Heroku or OpenShift, an artifact is unnecessary.)
6. Create a cloud run configuration for deploying your artifact or code to the cloud. (IntelliJ IDEA can create such a run configuration automatically.)
7. Execute the run configuration to deploy your artifact or code to the cloud.

Cloud integration plugins

There is a separate cloud integration plugin for each of the supported cloud platforms. All the cloud integration plugins are bundled with the IDE and enabled by default.

The plugin names, normally, include the cloud name and the word *integration*, e.g. *Heroku integration*.

See also, Enabling and Disabling Plugins .

Cloud run configurations

There are run configurations for each of the supported clouds:

- Run/Debug Configuration: CloudBees Deployment
- Run/Debug Configuration: Cloud Foundry Deployment
- Run/Debug Configuration: Google App Engine Deployment
- Run/Debug Configuration: Heroku Deployment
- Run/Debug Configuration: OpenShift Deployment

Getting Started with Heroku

This tutorial illustrates main tasks related to working with Heroku.

– Creating a Heroku user account
– Generating and installing SSH keys
– Making sure that Heroku and Git Integration plugins are enabled
– Registering your Heroku user account in IntelliJ IDEA
– Creating a project
– Exploring a run configuration
– Deploying your app
– Modifying the source code
– Redeploying the app
– Undeploying the app
– Disconnecting from Heroku

Creating a Heroku user account

Go to the Heroku web site and sign up for a user account.

Generating and installing SSH keys

To be able to deploy your code to the cloud, you have to upload your public SSH key to Heroku. You can do that, for example, when registering your Heroku user account in IntelliJ IDEA .

If you don't have a private/public SSH key pair, generate one. Search the Internet for corresponding tools and instructions.

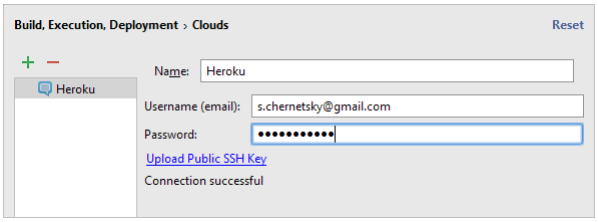
Put your keys to the `.ssh` folder in your user home directory.

Making sure that Heroku and Git Integration plugins are enabled

To be able to work with Heroku and Git on Heroku, the Heroku Integration and the Git Integration plugins must be enabled . To make sure that these plugins are enabled, use the Plugins page of the Settings/Preferences dialog (`Ctrl+Alt+S` | Plugins).

Registering your Heroku user account in IntelliJ IDEA

1. Open the Settings/Preferences dialog (e.g. `Ctrl+Alt+S`) and select Build, Execution, Deployment | Clouds .
2. Click + and select Heroku.
3. Specify your user name and password.

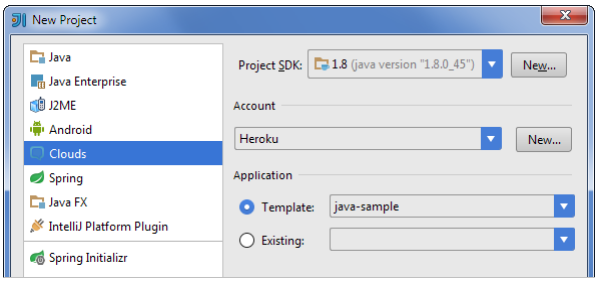


4. If you haven't uploaded your public SSH key to Heroku yet, you can do that now.
   Click Upload Public SSH Key and select the key file in the dialog that opens.

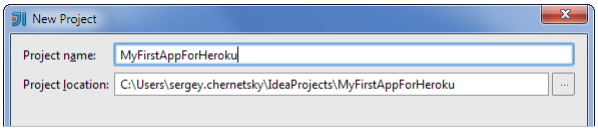
   The key file should have the `.pub` extension and may be called `id_rsa.pub` , `id_dsa.pub` or something similar.
5. Click OK.

Creating a project

1. Click Create New Project on the Welcome screen, or select File | New | Project on the main menu.
   The New Project wizard opens.
2. In the left-hand pane, select Clouds. The rest of the settings should be similar to this:



   Click Next.
3. Specify the name for your new project (e.g. `MyFirstAppForHeroku`).



   Click Finish.

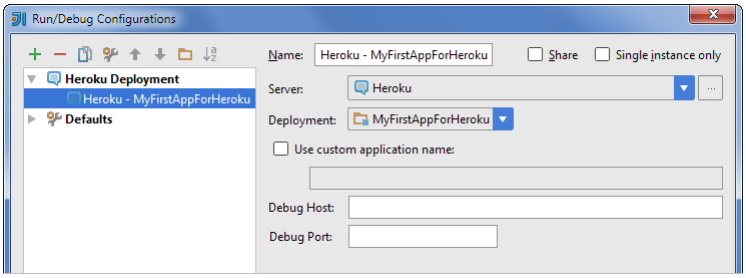
Exploring a run configuration

To deploy and debug your applications on Heroku, IntelliJ IDEA provides Heroku Deployment run/debug configuration .

There is already one such configuration in your project.

Let's take a quick look at its settings.

1. Select Run | Edit Configurations.



The run/debug configuration specifies that your module source code should be deployed to Heroku. (`MyFirstAppForHeroku` is the name of a module .)

The application will be deployed under its default name, in this case, `myfirstappforheroku` . (The application name defines its URL, `https://<app-name>.herokuapp.com/` .) If you want to use a different name, select the Use custom application name check box and specify the name.

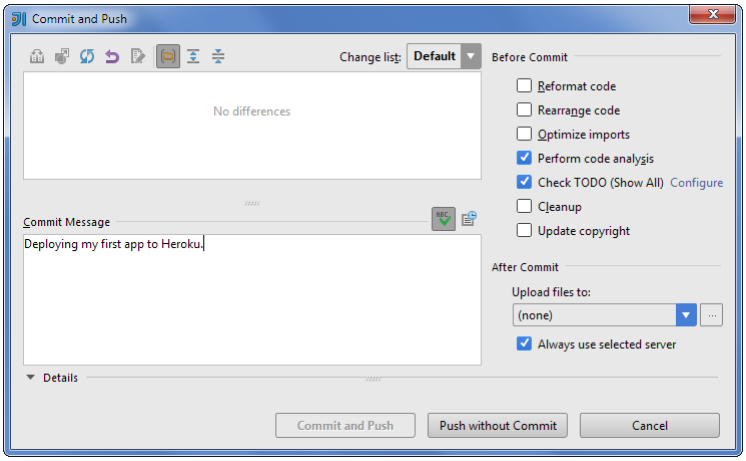
If you wanted to use the run configuration also for debugging your app, you'd specify the debug host and port.

To create another run configuration for Heroku, you should click + and select Heroku Deployment. For more information, see Working with Run/Debug Configurations and Run/Debug Configuration: Heroku Deployment .

2. Click OK.

Deploying your app

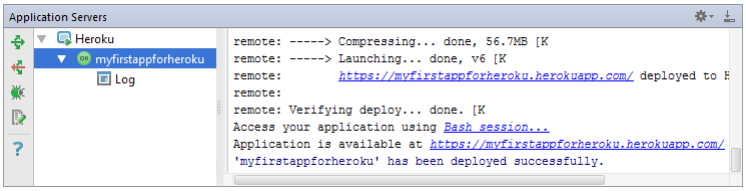
1. Start the run configuration: click ▶ to the right of the run configuration selector or press `Shift+F10` .
   The Commit and Push dialog opens. This dialog is used to commit changes to your local Git repository and to push them to a remote Git repository, in this case, the one on Heroku.
2. Write the commit message and click Push without Commit.



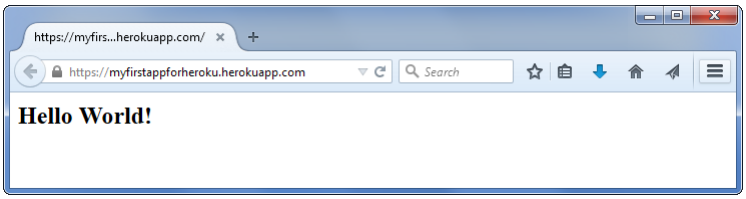
As a result (all the following takes place on Heroku; you can monitor the process in the Application Servers tool window):

- A Git repository for your app source code is created.
- The app dependencies specified in `pom.xml` are installed.
- Your app is built and started.

Finally, when your app is deployed, the link to it is shown within the line *Application is available at ...*



3. Click the link.
   Your web browser opens and your application output is shown.



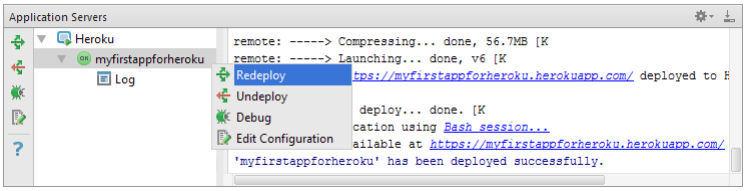
Modifying the source code

1. Open the file `index.jsp` for editing: select the file in the Project tool window and press F4.
2. Change the text, say, to `Hello from IntelliJ IDEA!`

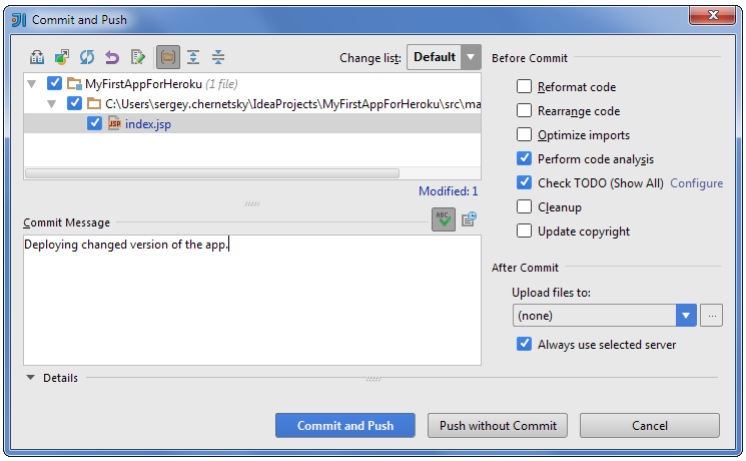
Redeploying the app

To publish the changed version of your app, you should redeploy the app.

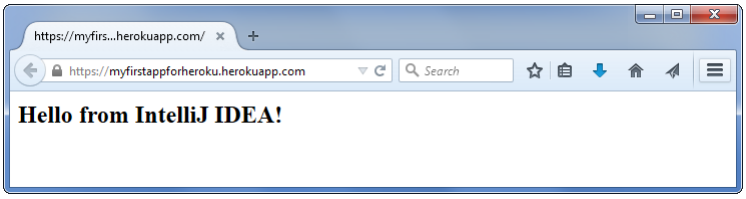
1. In the Application Servers tool window, right-click your app and select Redeploy.



2. In the Commit and Push dialog, write the commit message and click Commit and Push.



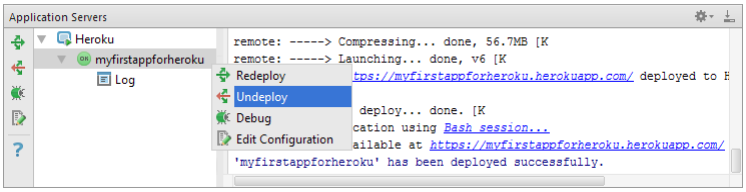
3. When the message *'<app-name>' has been deployed successfully* is output, switch to the web browser and reload the page to see the changes.



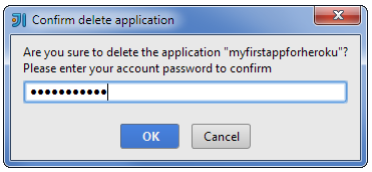
Undeploying the app

When your app is no longer needed, you should undeploy it (i.e. remove it from Heroku along with its source code).

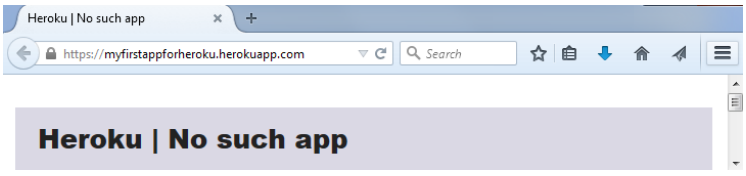
1. In the Application Servers tool window, right-click your app and select Undeploy.



2. Provide your Heroku password to confirm your intention to delete the app.

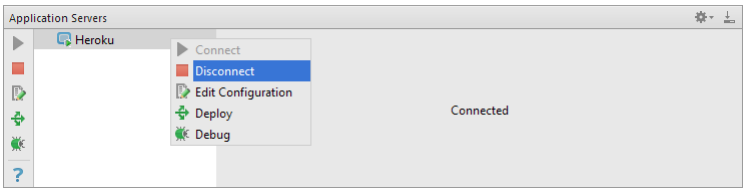


3. Switch to the web browser and reload the page to see that you app has become unavailable.



Disconnecting from Heroku

- In the Application Servers tool window, right-click Heroku and select Disconnect.



Getting Started with Google App Engine

IntelliJ IDEA lets you create Google App Engine projects and upload your applications to Google infrastructure.

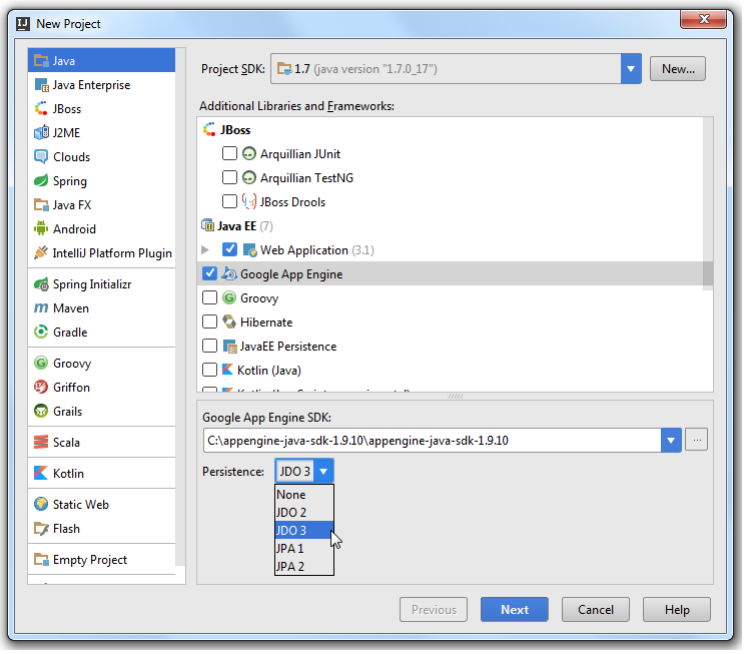
– Before you start
– Creating Google App Engine Project
– Checking Project Structure
– Running the Application
– Debugging the Application
– Configuring Google App Account
– Deploying Google App Engine Application
– Using Google App Engine Deployment

Before you start

Before you start creating your Google App Engine project, make sure that the Google App Engine plugin is downloaded and enabled in IntelliJ IDEA.

Creating Google App Engine Project

1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen. Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.
2. In the left-hand pane, select Java.
3. In the right-hand pane, select your project SDK.
4. Under Additional Libraries and Frameworks select Google App Engine.

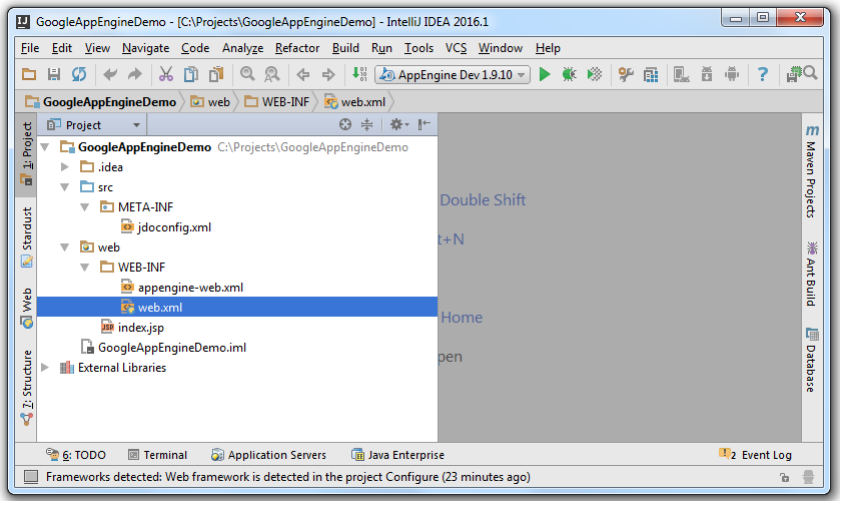


Note that the Web Application option will be selected automatically.

5. In the Google App Engine SDK field, select the SDK you want to use. If the list is empty click Download link to download the latest Google App Engine SDK. If the field doesn't contain the SDK that you want, click ... and select the installation folder of the required Google App Engine SDK in the dialog that opens .
6. Click Next.
7. On the next page of the wizard, specify the name and location settings.

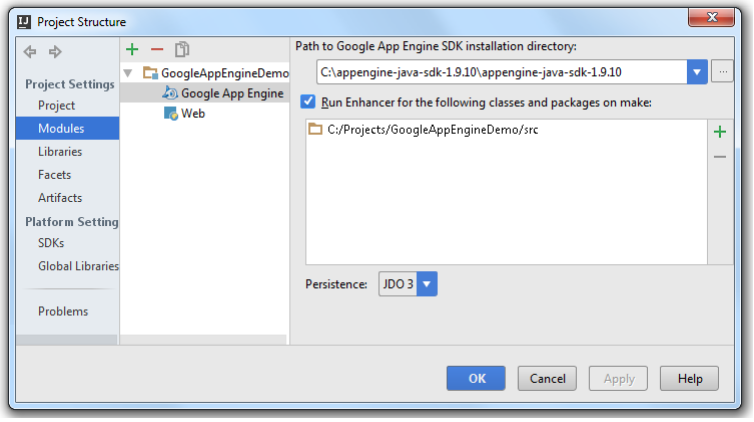
   For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

   IntelliJ IDEA creates an empty Google App Engine project. Depending on the selected persistence type, IntelliJ IDEA generates persistence.xml for JPA or jdoconfig.xml for JDO and takes the required libraries from the App Engine SDK.



Checking Project Structure

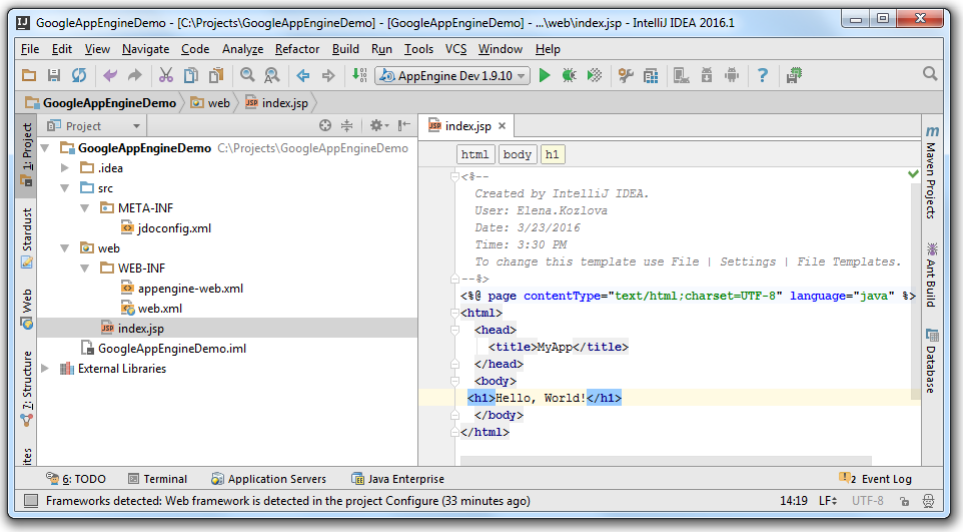
1. On the main menu, select Project Structure | Modules.



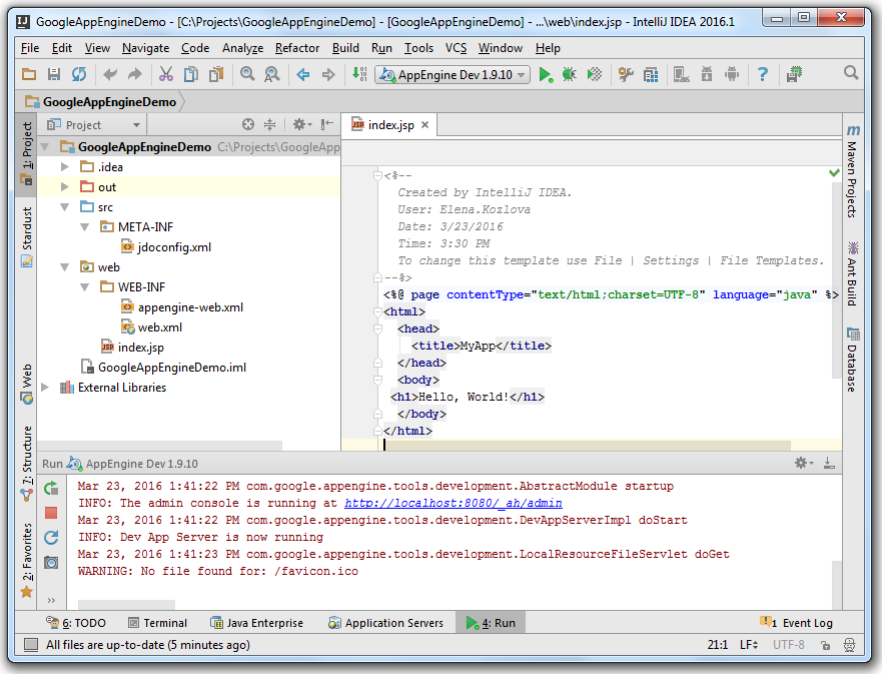
2. Check settings for the Google App Engine facet to make sure everything was configured properly.

Running the Application

1. Enter your code or (like in our case) you can modify the index.jsp file.



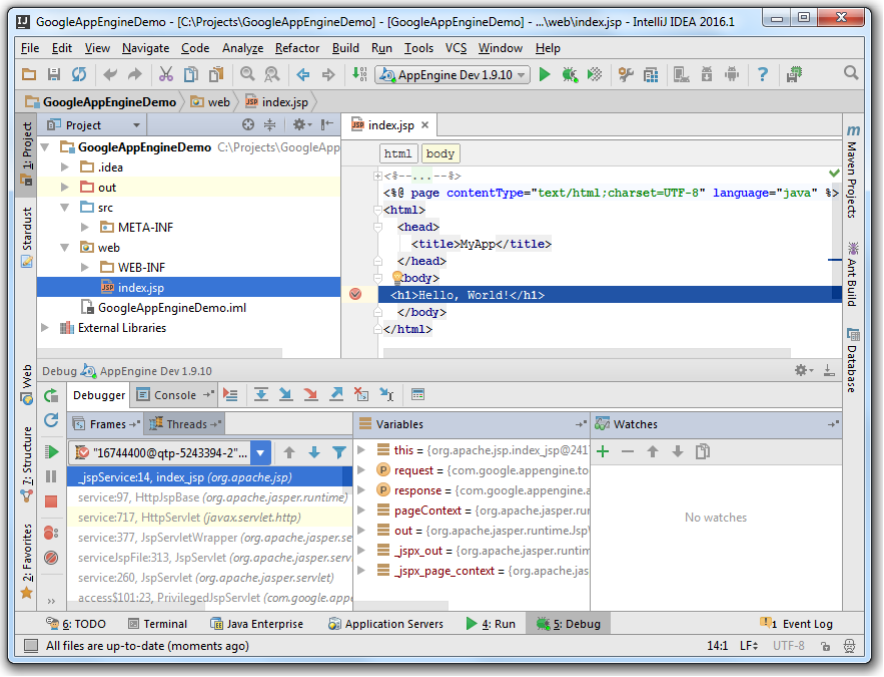
2. Press icon to run the application.



3. View the result in the default browser.

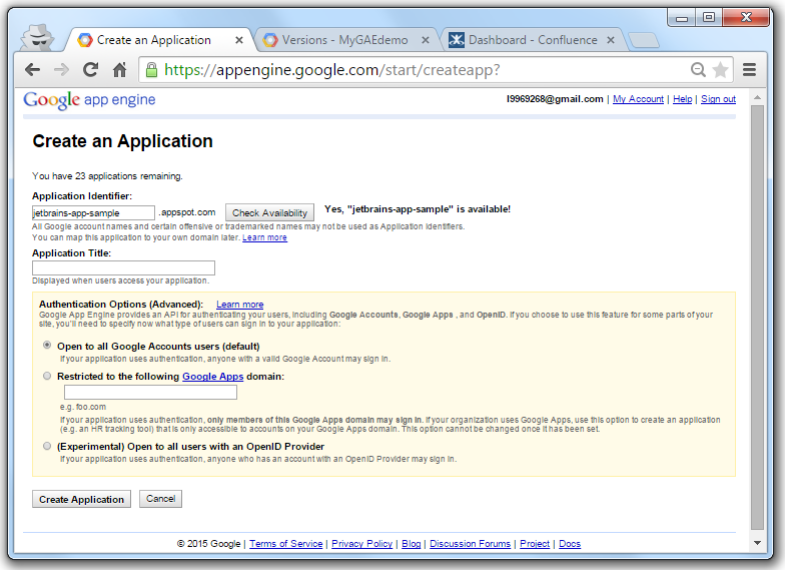
Debugging the Application

1. On the main menu, select Run|Debug 'AppEngine Dev' or click the toolbar button.
2. View the results in the Debugger tool window.

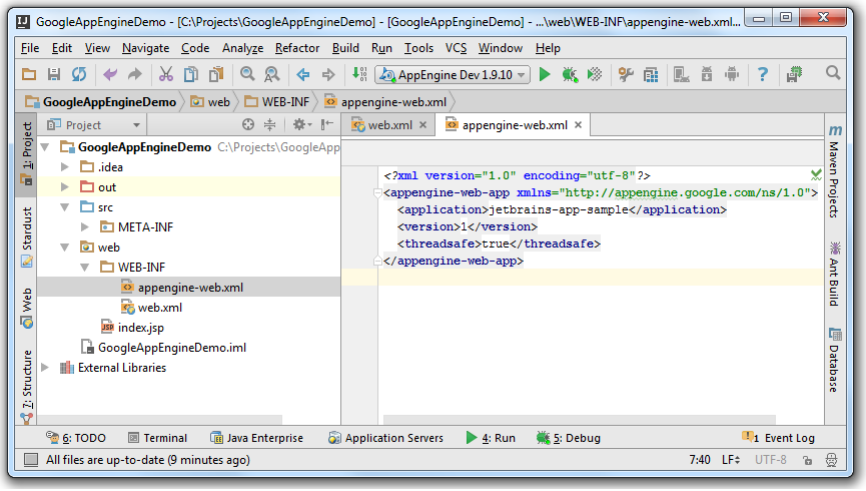


Configuring Google App Account

1. If you going to upload your application to the cloud or deploy the artifact on the cloud server, create an account at Google App Engine .

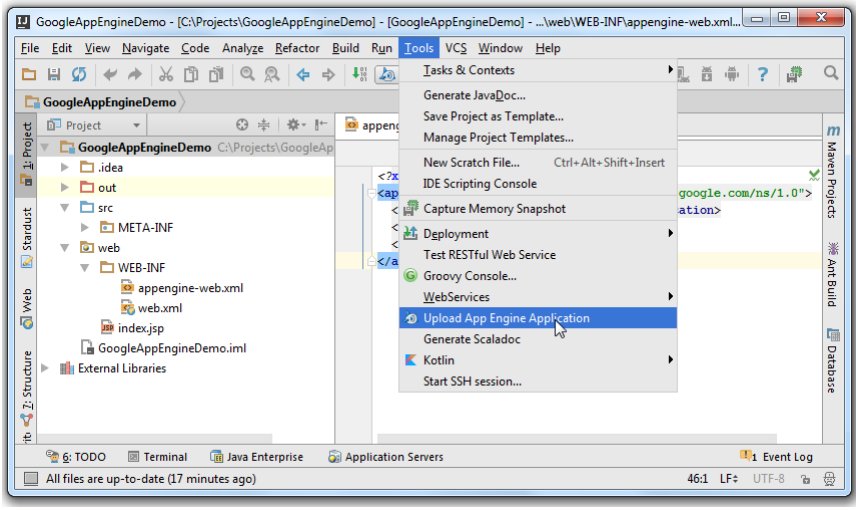


2. Add the name of the created application to the web.xml file.

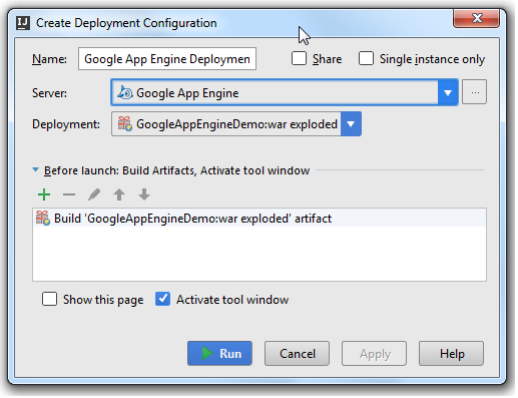


Deploying Google App Engine Application

1. On the main menu, select Tools | Upload App Engine Application.

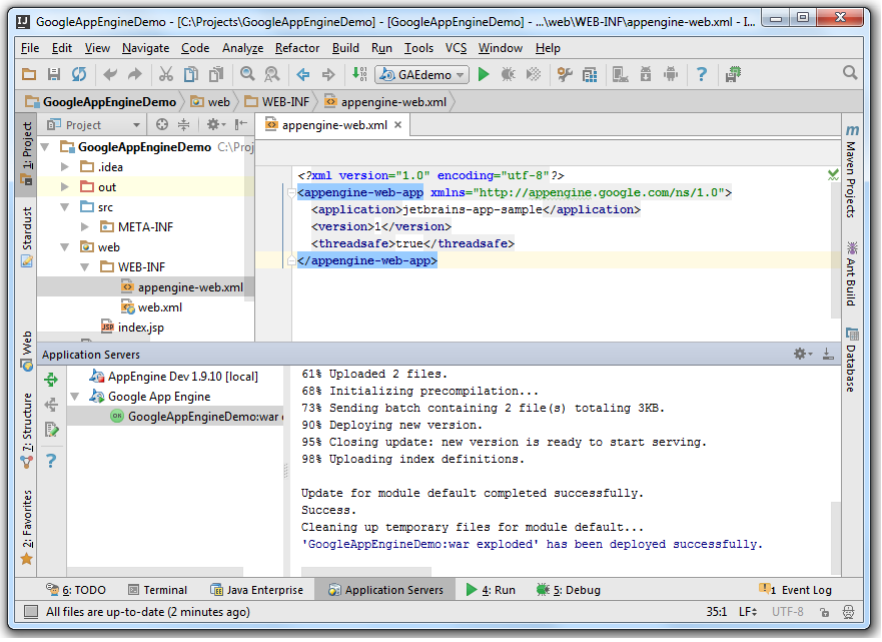


2. In the dialog that opens specify the deployment configuration settings and click Run.



3. During the deployment IntelliJ IDEA might prompt you for your Google Account credentials.

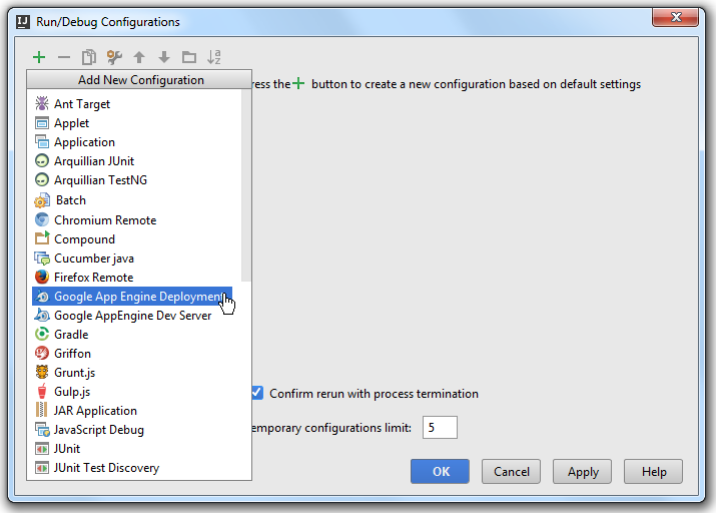
   IntelliJ IDEA displays the progress in the Application Servers tool window.



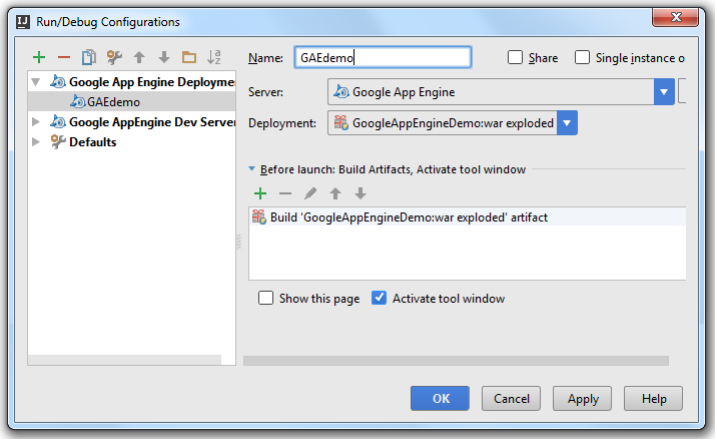
4. View the result in your default browser.

Using Google App Engine Deployment

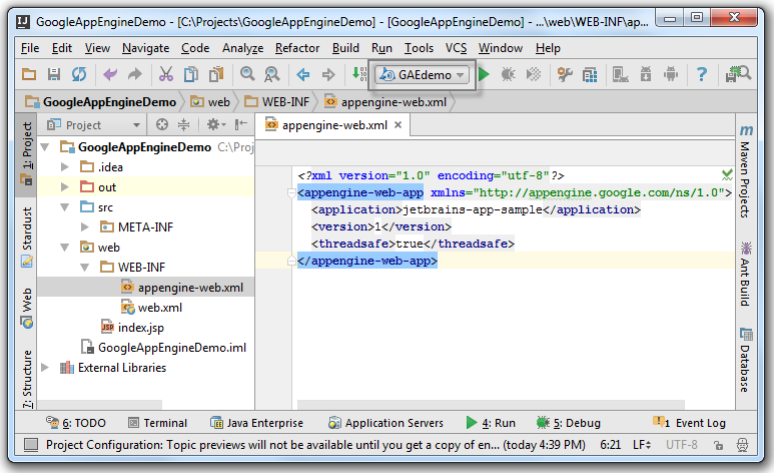
1. Select Run | Edit Configurations.
2. In the Run/Debug Configurations dialog, select + to add a new configuration.
3. From the drop-down list select Google App Engine Deployment.



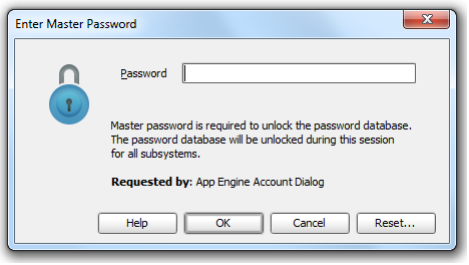
4. On the right-hand side specify the Google App Engine Deployment settings and click OK.



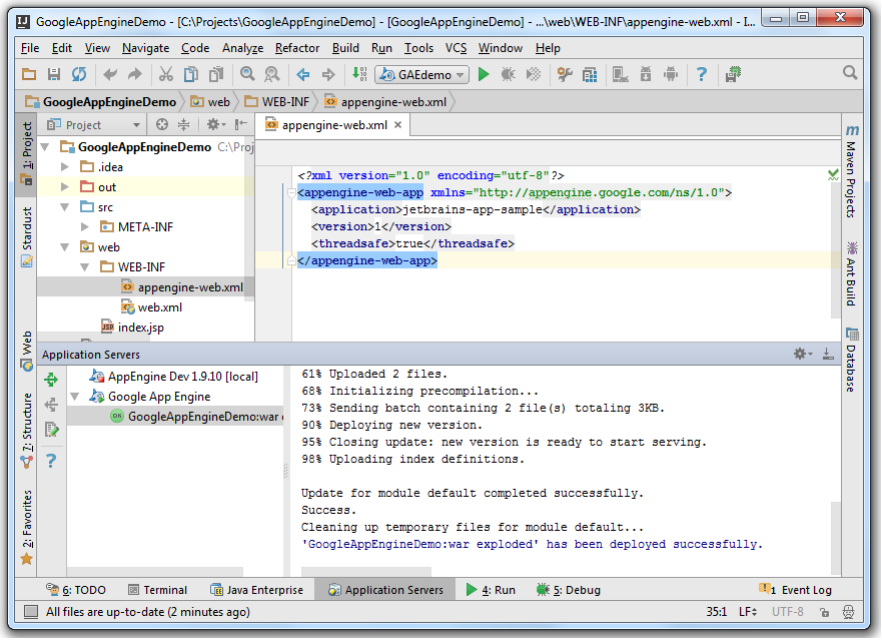
5. The name of the configuration appears on the main tool bar, click ▶ to start deploying.



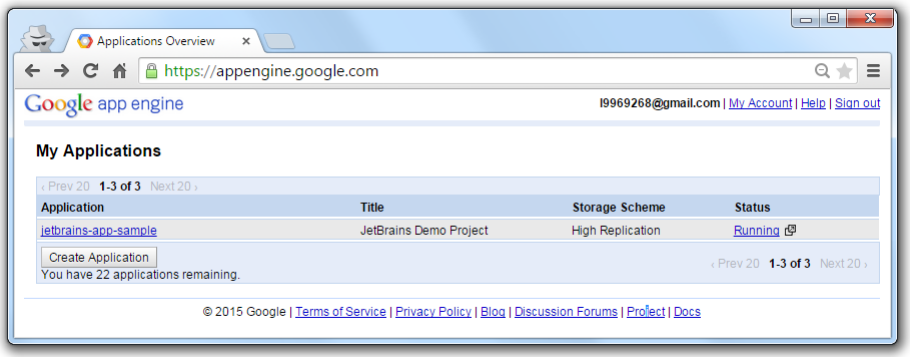
Note that the IntelliJ IDEA might ask for your Google Account password.



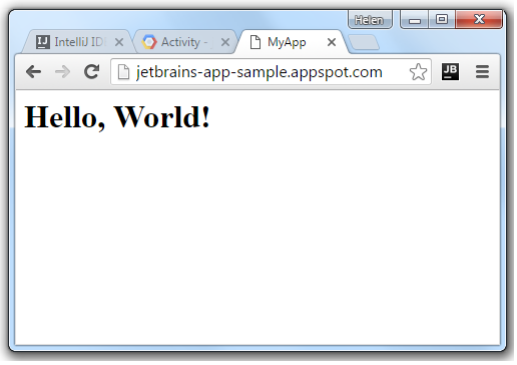
6. IntelliJ IDEA displays the progress of deployment in the Application Servers tool window.



7. Open Google App Engine page.



8. On the Application Overview page, click Running next to your application name to see the result of your deployment.



Tutorial: Developing a Java EE application

This tutorial illustrates the Java EE application development workflow.

The application that we will develop will be a minimal one. It'll be a one JSP page Java web application. However, the IntelliJ IDEA features discussed here are applicable to Java EE applications of any complexity.

- Before you start
- Creating a project
- Exploring the project structure
- Developing source code
- Running the application
- Modifying the code and observing the changes
- Exploring a run configuration. Fixing debug settings
- Exploring an artifact configuration
- Packaging the application into a WAR file
- Deploying an artifact onto a running server

Before you start

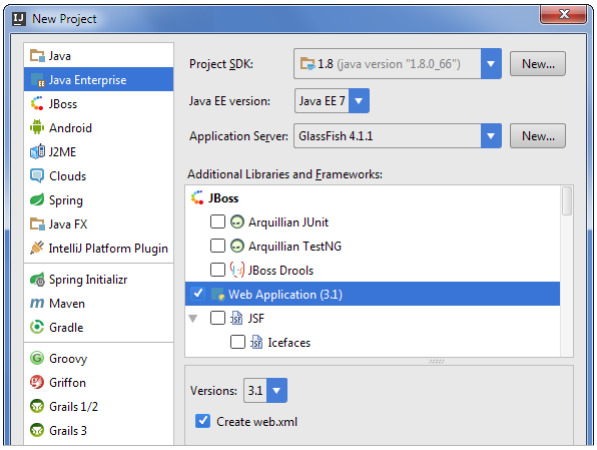
Make sure that the following software is installed on your computer:

- IntelliJ IDEA ULTIMATE Edition.
- Java SE Development Kit (JDK), version 6 or later. Download Oracle JDK .
- GlassFish Server, version 3.0.1 or later. Download GlassFish . (You can use any other Java EE-enabled application server. GlassFish is used here just as an example.)
- A web browser.

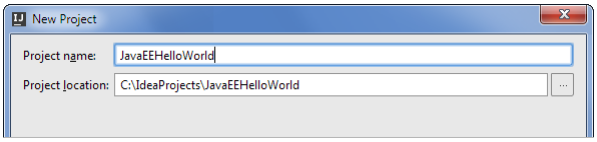
Creating a project

1. Click Create New Project on the Welcome screen, or select File | New | Project.
   The New Project wizard opens.
2. In the left-hand pane, select Java Enterprise.
3. If the JDK that you want to use is already defined in IntelliJ IDEA, select that JDK from the Project SDK list. Otherwise, click New, select JDK, and select the JDK installation folder in the dialog that opens.
4. Specify the application server that you are going to use. (We'll use GlassFish Server.)
   If GlassFish is not defined in IntelliJ IDEA yet, click New to the right of the Application Server field and select Glassfish Server.

   In the Glassfish Server dialog, specify the GlassFish Server installation directory.
5. Under Additional Libraries and Frameworks, select the Web Application check box.



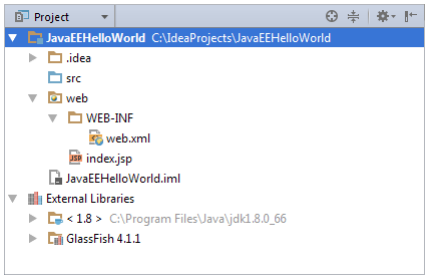
   Click Next.
6. Specify the name for your new project (e.g. `JavaEEHelloWorld`).



   Click Finish and wait while IntelliJ IDEA is creating the project.

Exploring the project structure

When the project is created, you'll see something similar to this in the Project tool window.



– JavaEEHelloWorld is a module folder (which in this case coincides with the project folder). The `.idea` folder and the file `JavaEEHelloWorld.iml` contain configuration data for your project and module respectively. The folder `src` is for your Java source code. The folder `web` is for the web part of your application. At the moment this folder contains the deployment descriptor `WEB-INF\web.xml` and the file `index.jsp` intended as a starting page of your application.

– External Libraries include your JDK and the JAR files for working with GlassFish.

Developing source code

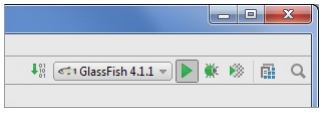
Our application will be a single JSP page application. Its only function will be to output the text *Hello World!*

1. Open `index.jsp` for editing: select the file in the Project tool window and press `F4`.
2. Between `<body>` and `</body>` type `Hello, World!`

The code at this step is ready.

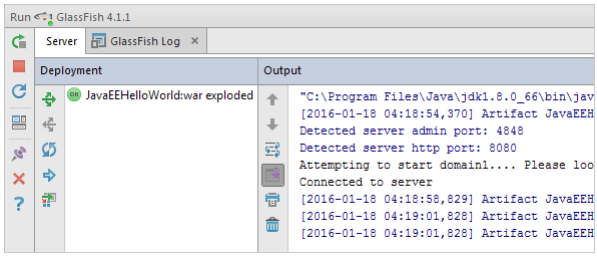
Running the application

In the upper-right part of the workspace, click ▶.

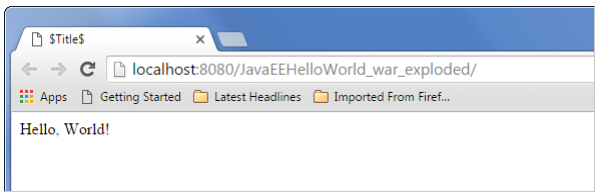


IntelliJ IDEA compiles your source code and builds an application artifact. (We'll discuss artifacts a bit later. For now, the artifact is your application in deployment-ready format.)

After that, the Run tool window opens. IntelliJ IDEA starts the server and deploys the artifact onto it.



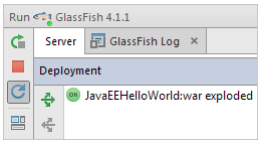
Finally, your default web browser starts and you see the application output `Hello, World!` there.



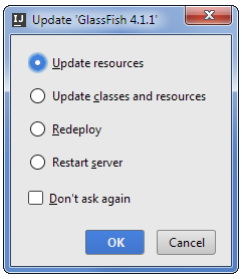
Modifying the code and observing the changes

1. In `index.jsp`, change `Hello, World!` to `Hello!`.

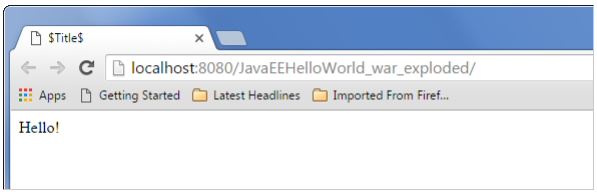
2. In the Run tool window, click Update.



3. In the Update dialog, select Update resources and click OK. (For more information, see Application update options.)



4. Switch to the web browser and reload the page to see the changes.



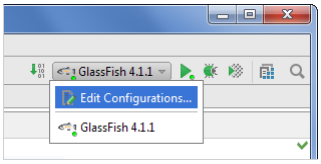
See also, Updating Applications on Application Servers .

Exploring a run configuration. Fixing debug settings

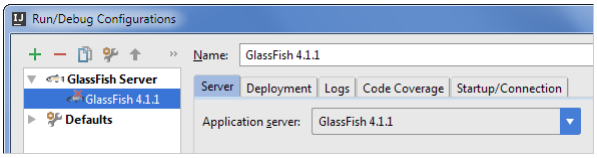
When creating the project, we specified GlassFish as an application server. As a result, IntelliJ IDEA created a run configuration for GlassFish.

When we performed the Run command ( ), we started that run configuration. Now let's take a look at the run configuration and see how its settings map onto the events that we've just observed.

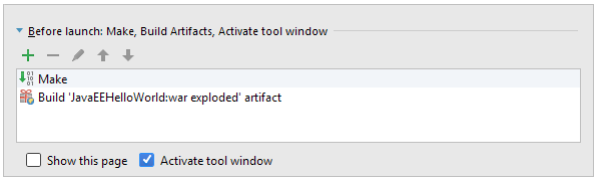
1. Click the run configuration selector and select Edit Configurations.



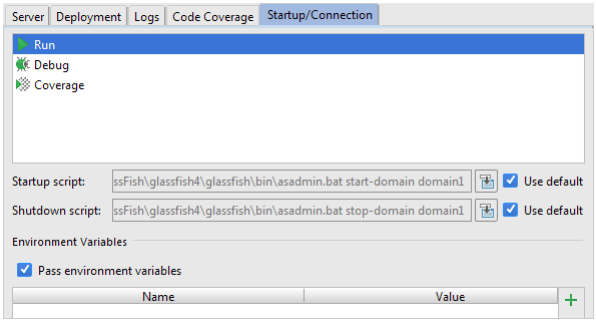
The Run/Debug Configurations dialog opens and the settings for the GlassFish run configuration are shown.



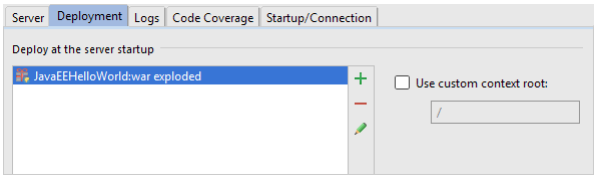
The Before launch task list (in the lower part of the dialog) specifies that the application code should be compiled and the corresponding artifact should be built prior to executing the run configuration.



2. Select the Startup/Connection tab to see how the server is started in the run, debug and code coverage modes.

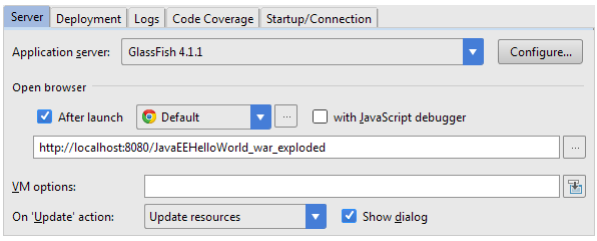


3. Select the Deployment tab to see which artifacts are deployed after the server is started.

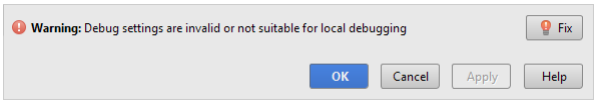


4. Go back to the Server tab.
   The settings under Open browser specify that after launch (i.e. after the server is started and the artifacts are deployed onto it) the default web browser should start and go to the specified URL (`http://localhost:8080/JavaEEHelloWorld_war_exploded`).

   The settings to the right of On 'Update' action specify that on clicking in the Run tool window the Update dialog should be shown and the Update resources option should be used by default. (The last used update option becomes the default one).



5. Note the warning *Debug settings are invalid* in the lower part of the dialog. Click Fix. (IntelliJ IDEA makes the necessary changes to GlassFish `domain.xml` configuration file.)



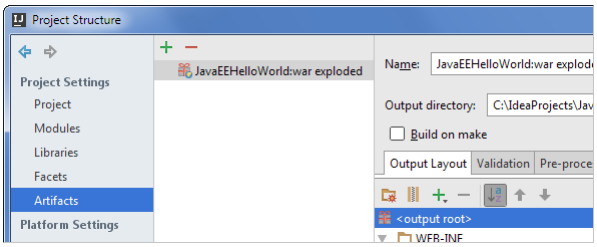
6. Click OK.

Exploring an artifact configuration

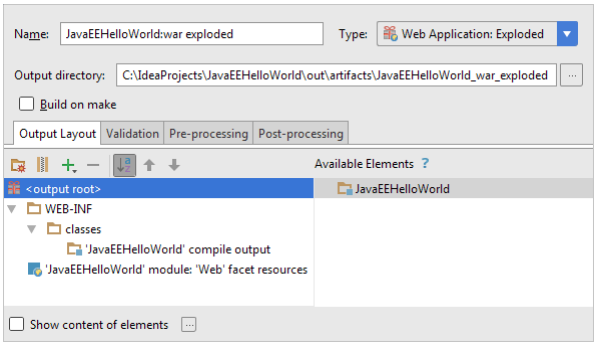
The word artifact in IntelliJ IDEA may mean one of the following: 1) an artifact configuration, i.e. a specification of the output to be generated for a project; 2) an actual output generated according to such a specification (configuration).

When creating the project, we indicated that we were going to develop a web application. As a result, IntelliJ IDEA, among other things, created a configuration for building a web application. Let's have a look at this configuration.

1. Open the Project Structure dialog: File | Project Structure or `Ctrl+Shift+Alt+S` .
2. Under Project Settings, select Artifacts.
   The available artifact configurations are shown in the pane to the right under + and –. (There's only one configuration at the moment.)



The artifact settings are shown in the right-hand part of the dialog.



Type. The artifact type is Web Application: Exploded. This is a decompressed web application archive (WAR), a directory structure that is ready for deployment onto a web server.

Output directory. The artifact, when built, is placed into `<project_folder>\out\artifacts\JavaEEHelloWorld_war_exploded` .

Output Layout. The artifact structure is shown in the left-hand pane of the Output Layout tab.

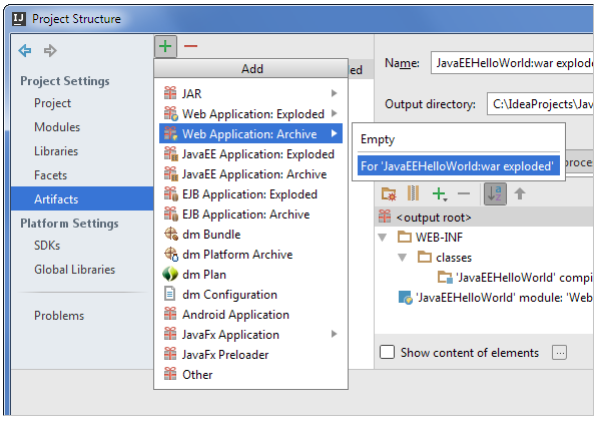
The <output root> corresponds to the output directory. Other elements have the following meanings:

- 'JavaEEHelloWorld' compile output represents compiled Java classes whose sources are located in the `src` directory. These are placed into `WEB-INF\classes` in the output directory.
- 'Web' facet resources represent the contents of the `web` directory.

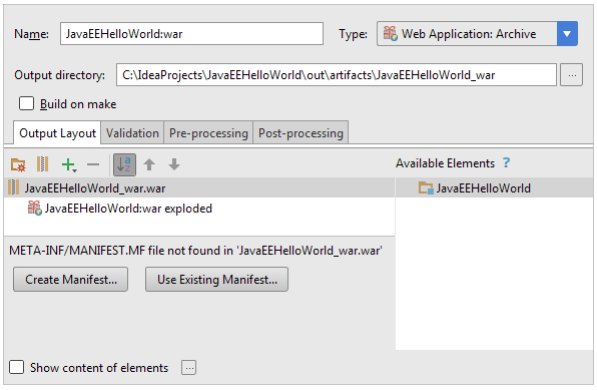
Packaging the application into a WAR file

When you get to the stage when you are happy with your application, you may want to place it in a WAR (web application archive). To do that, you should create an appropriate artifact configuration and then build the artifact:

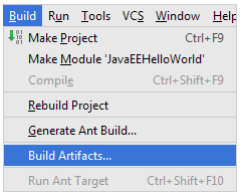
1. Click +, point to Web Application: Archive and select For 'JavaEEHelloWorld: war exploded'.



A new artifact configuration is created and its settings are shown in the right-hand part of the dialog.

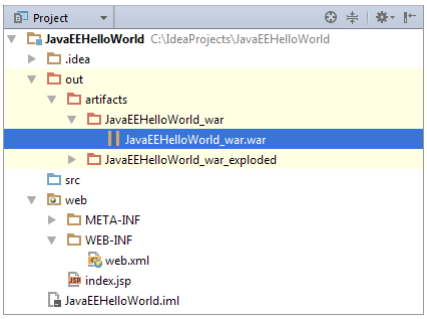


2. If necessary, create a manifest file for your archive: click Create Manifest and agree to the location suggested by IntelliJ IDEA (`web\META-INF\MANIFEST.MF`).
3. Click OK in the Project Structure dialog.
4. Select Build | Build Artifacts.



5. In the Build Artifact popup, point to JavaEEHelloWorld:war and select Build.

Now if you look at the `out\artifacts\JavaEEHelloWorld_war` folder, you'll see the archive there.



Deploying an artifact onto a running server

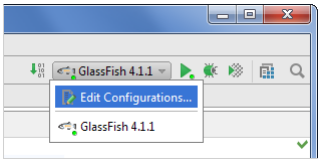
Sometimes you need to deploy your app onto a running server. This section provides a how-to example.

Server run configurations that don't start a server are called *remote*. Such run configurations can be used, for example, for deploying applications to servers that are already running. (See Local and remote run configurations.)

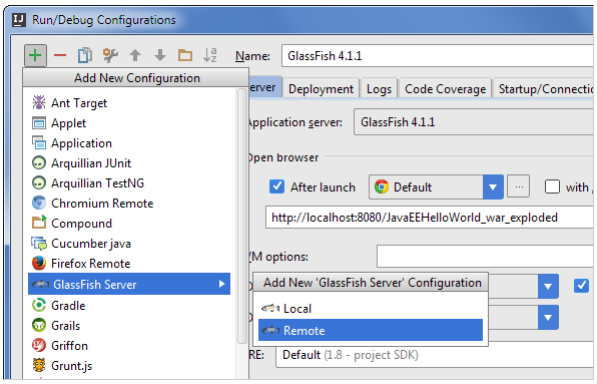
Let's create a run configuration for deploying our WAR artifact to the running server and see how it works. (By now, the server has been started by the run configuration discussed earlier .)

Click the run configuration selector and select Edit Configurations

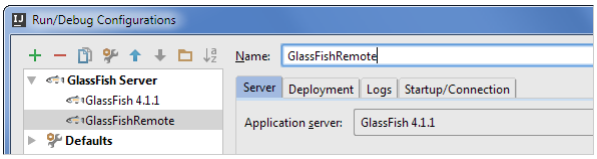
1. Click the run configuration selector and select Edit Configurations:



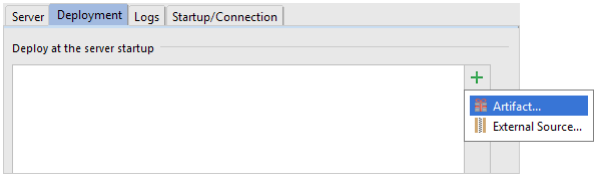
2. Click +, point to GlassFish Server and select Remote.



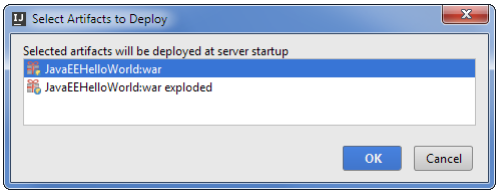
3. Change the run configuration name `Unnamed` to something more sensible (e.g. `GlassFishRemote`).



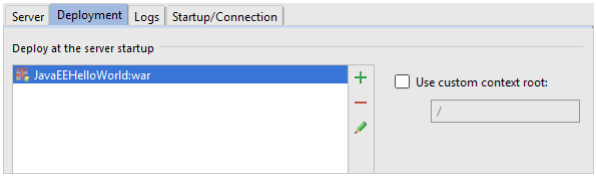
4. Specify the artifact to be deployed to the server: select the Deployment tab, click + and select Artifact.



In the dialog that opens, select the WAR artifact.



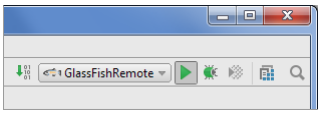
The result should look similar to this:



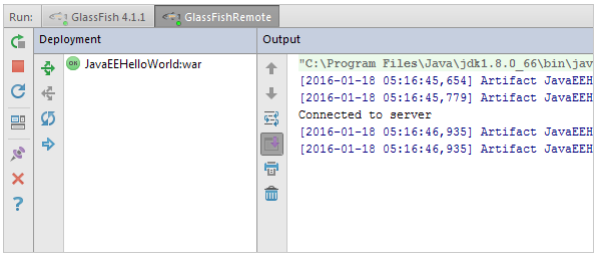
5. Click OK in the Run/Debug Configurations dialog.

   Now let's see how this run configuration works.

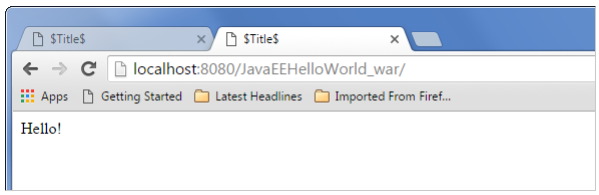
6. Execute the run configuration: click ▶.



The run configuration output is shown in the Run tool window.



After a while, a new tab in your web browser opens, and you see the application output there.


$Title$
$Title$
localhost:8080/JavaEEHelloWorld_war/
Apps
Getting Started
Latest Headlines
Imported From Firef...
Hello!

JavaFX

JavaFX support in IntelliJ IDEA includes code completion, search, navigation and refactoring in JavaFX-specific source files (including FXML and JavaFX CSS files), integration with JavaFX Scene Builder, JavaFX application packaging capabilities, and more.

- Preparing for JavaFX Application Development
- Developing a JavaFX Hello World Application: Coding Examples
- Opening FXML Files in JavaFX Scene Builder
- Packaging JavaFX Applications
- Applications with a Preloader: Project Organization and Packaging

Preparing for JavaFX Application Development

On this page:

- Preparing to develop JavaFX applications
- Enabled or disabled JavaFX plugin ?
- Defining JDK 7 in IntelliJ IDEA
- Specifying the path to the JavaFX Scene Builder executable
- Creating a project for JavaFX development
- Exploring project
- Running the sample application

Preparing to develop JavaFX applications

To prepare for JavaFX application development, follow these general steps:

1. Download and install JDK 7 or a later version (earlier JDK versions don't include the JavaFX SDK necessary for JavaFX application development).
2. If you are going to use JavaFX Scene Builder , download and install it as well.
3. Make sure that the JavaFX plugin is enabled. (JavaFX support in IntelliJ IDEA in based on the JavaFX plugin. This plugin is bundled with the IDE and enabled by default.) See To make sure that the JavaFX plugin is enabled.
4. Define the JDK in IntelliJ IDEA. You can do that separately (see To define JDK 7 in IntelliJ IDEA) or when creating a project or module (see To create a project for JavaFX application development from scratch).
5. If necessary, specify the path to the JavaFX Scene Builder executable file. If you do so, you'll be able to open your FXML files in the Scene Builder right in IntelliJ IDEA. See To specify the path to the JavaFX Scene Builder executable file.
6. Create a project for your JavaFX application development. Your can create the corresponding project from scratch or, if you already have the source files you want to continue working with, you can create a project by importing the corresponding sources. See To create a project for JavaFX application development from scratch or Creating a Project by Importing Existing Sources . See also, Using Scene Builder with IntelliJ IDEA .

Enabled or disabled JavaFX plugin ?

Even though the JavaFX plugin is enabled by default, it's always worth making sure that this plugin is still enabled before you start developing a JavaFX application.

To make sure that the JavaFX plugin is enabled

1. Open Settings/Preferences dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, select Plugins.
3. In the right-hand part of the dialog, on the Plugins page , type `fx` in the search box. As a result, only the plugins whose names and descriptions contain `fx` are shown in the list of plugins.
4. If the check box to the right of JavaFX is not selected, select it.
5. Click OK in the Settings/Preferences dialog.
6. If suggested, restart IntelliJ IDEA.

Defining JDK 7 in IntelliJ IDEA

Once you have JDK 7 or a later version downloaded and installed, you should define it in IntelliJ IDEA. You can do that separately, the way described in this section. You can also do that at a later time, when creating a project for your JavaFX application development.

To define JDK 7 in IntelliJ IDEA

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the leftmost pane, under Platform Settings, click SDKs.
3. Above the pane to the right, click + and select JDK.
4. In the dialog that opens , select the JDK installation directory and click OK.
5. Click OK in the Project Structure dialog.

Specifying the path to the JavaFX Scene Builder executable

To be able to open your FXML files in JavaFX Scene Builder right in IntelliJ IDEA, you should specify where the Scene Builder executable file is located. You can do that separately, the way described in this section. You can also do that at a later time, the first time you open an FXML file in the Scene Builder from within IntelliJ IDEA.

To specify the path to the JavaFX Scene Builder executable file

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, select Languages and Frameworkds | JavaFX.
3. In the right-hand part of the dialog, on the JavaFX page, click ... to the right of the Path to SceneBuilder field.
4. In the dialog that opens , select the Scene Builder executable file and click OK.

5. Click OK in the Settings/Preferences dialog.

See also, Opening FXML Files in JavaFX Scene Builder .

Creating a project for JavaFX development

To create a project for JavaFX application development from scratch

1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen . Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.

2. On the first page of the wizard , in the left-hand pane, select JavaFX.
   In the right-hand part of the page, specify the SDK (JDK) to be used in your project.

   Select the JDK from the list, or click New, select JDK and select the installation folder of the desired JDK.

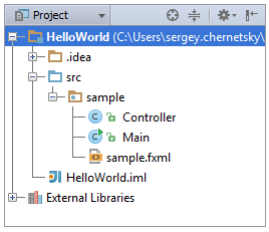
   Note that the JDK version 7 or later should be specified.

   Click Next.

3. Specify the project name and location, and click Finish.

Exploring project

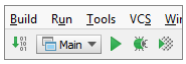
Let's take a quick look at what we've got in the project.



The folder `src` is for your source code. In this folder, there is already a package called `sample` containing three files:

– `Main.java.` This is the main application class for starting the sample application.
– `sample.fxml.` This is the FXML file for defining the user interface.
– `Controller.java.` This is the controller class intended to handle user interactions with the UI.

In addition to the sample application source code, there is a run/debug configuration for running or debugging the application. The run configuration has the same name as the main application class (`Main`) and is shown in the run configuration selector on the toolbar.



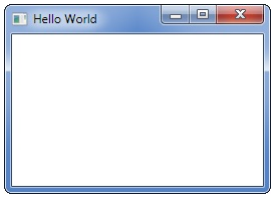
There is also an artifact configuration intended for packaging your application. (This artifact configuration is not visible at the moment). We'll discuss this configuration later (see Packaging JavaFX Applications).

Running the sample application

To make sure that everything is fine with the project, let's run the sample application straight away:

– Click ▶ on the toolbar.

IntelliJ IDEA compiles the source code and then starts the application. The application window appears which, at the moment, is empty.



Close the application window.

Developing a JavaFX Hello World Application: Coding Examples

In this topic, we transform the sample application created by IntelliJ IDEA into a very basic JavaFX Hello World application. In this way, we show basic coding assistance features provided by the IDE. (The sample application is created by IntelliJ IDEA automatically when you create a project for your JavaFX application development from scratch, see To create a project for JavaFX application development from scratch.)

– Renaming the Controller class

– Developing the user interface

– Completing the code for the SampleController class

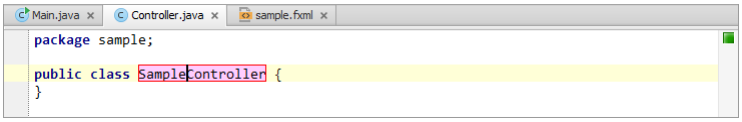
– Running the application

– Styling the UI with CSS

Renaming the Controller class

To adjust the sample application to your needs, you may want to start by renaming the files. To see how, let's perform the Rename refactoring for the class `Controller`. We'll rename this class to `SampleController`. You can use a different name if you like.

1. In the editor, place the cursor within the class name and select Refactor | Rename (alternatively, press `Shift+F6`).
2. Place the cursor in front of `Controller` and type `Sample`.



3. Press `Enter` to indicate that you have completed the refactoring.

Now, switch to `sample.fxml` in the editor and note that the value of the `GridPanel` `fx:controller` attribute has changed to `"sample.SampleController"`. (Initially it was `"sample.Controller"`.)

In a similar way you can change the names of other files if necessary.

Developing the user interface

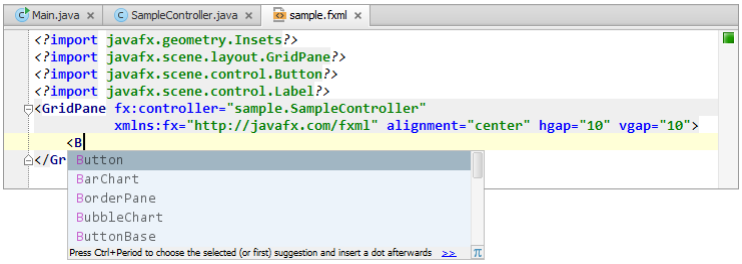
To show you how IntelliJ IDEA can help you write your code, let's implement a kind of Hello World JavaFX application.

In the user interface (UI), we'll define a button which when clicked will display the text `Hello World!` To do that, we'll add the following two elements between the opening and closing `<GridPane>` tags in the file `sample.fxml`:

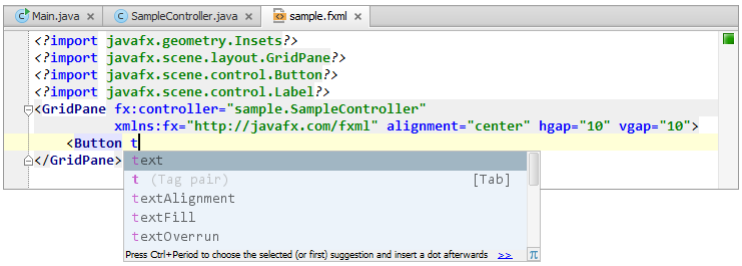
```
<Button text=
                    "Say 'Hello World'" onAction=
                    "#sayHelloWorld"/>
                    <Label GridPane.rowIndex=
                    "1" fx:id=
                    "helloWorld"/>
```

We suggest that you do everything by typing to see how code completion works.

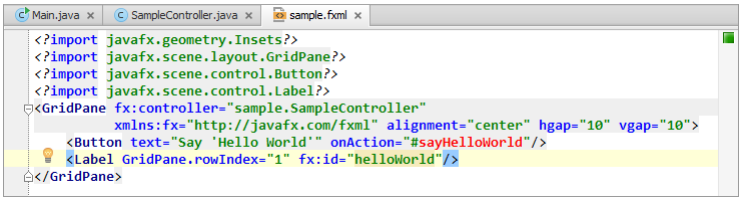
1. Go to the end of the opening `<GridPane>` tag and press `Enter` to start a new line.
2. Type `<B` and select Button.



3. Type space, type `t`, and select text.



4. In a similar way, add the remaining code fragments. The resulting code will look something similar to this:



As you see, `sayHelloWorld` is shown red and `helloWorld` is also highlighted. This means that IntelliJ IDEA cannot resolve the corresponding references.

To resolve the issues, let's use the quick fixes suggested by IntelliJ IDEA.

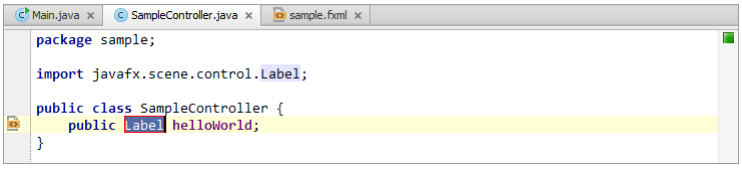
(In IntelliJ IDEA, it's a standard coding practice when you reference a field, method or class that doesn't yet exist and then use a quick fix to create the corresponding field, method or class.)

Completing the code for the SampleController class

Now we are going to define the field `helloWorld` in the `SampleController` class. We will also add the corresponding event handler method (`sayHelloWorld`) that will set the text for the `helloWorld` label. When doing so, as already mentioned, we'll use the quick fixes suggested by IntelliJ IDEA.

1. In `sample.fxml` , place the cursor within `helloWorld` . Click the yellow light bulb or press `Alt+Enter` .
2. Select Create Field 'helloWorld'.

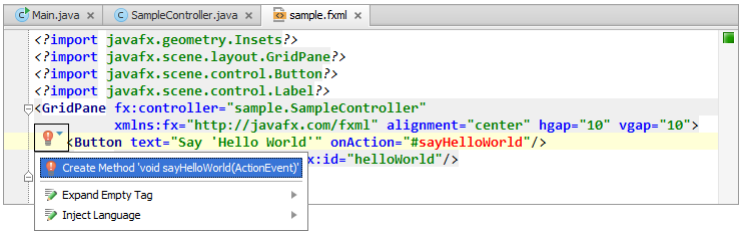
IntelliJ IDEA switches to `SampleController.java` where the declaration of the field `helloWorld` has been added.



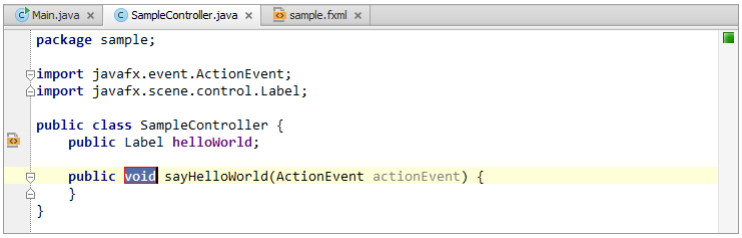
Note the red border around `Label` . You can edit the field type right away. We are not going to do that now, so press `Enter` to quit the refactoring mode.

Also note the import statement that has just been added (`import javafx.scene.control.Label;`) and the icon to the left of the field declaration. This is a navigation icon; click it to go back to `sample.fxml` .

3. Place the cursor within `sayHelloWorld` and press `Alt+Enter` .
4. Select Create Method 'void sayHelloWorld(ActionEvent)'.

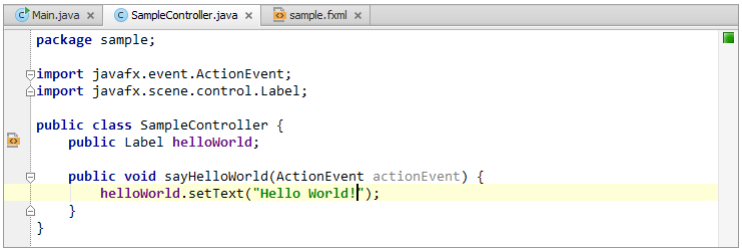


The corresponding method declaration is added to `SampleController.java` .



5. Press `Shift+Enter` to quit the refactoring mode and start a new line.
6. Type the following to set the text for the label:

```
helloWorld.setText("Hello
                   World!");
```



At this step, the code of the application is ready. Let's run the application to see the result.

Running the application

1. To run the application, click ▶ on the toolbar or press `Shift+F10`.
   The application window now contains the Say 'Hello World' button.

2. Click this button to see that the text Hello World! is shown.

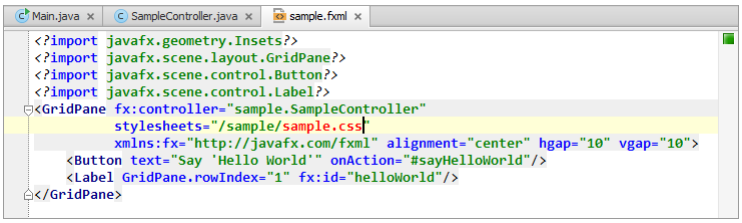
3. Close the application window.

Styling the UI with CSS

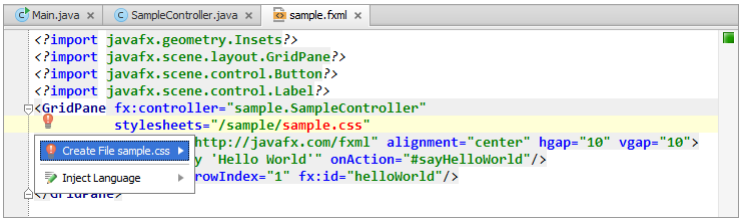
To complete the coding examples, let's change the appearance of the UI by adding a stylesheet and defining a couple of formatting styles in it.

1. In the file `sample.fxml`, add a reference to a (non-existing) CSS file `sample.css`. One way to do that is to add the `stylesheets` attribute within the opening `<GridPane>` tag, e.g.

```
stylesheets=
                    "/sample/sample.css"
```



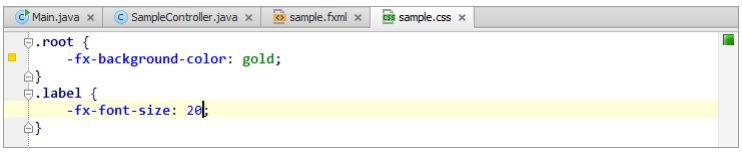
2. As before, use a quick fix to create the CSS file.



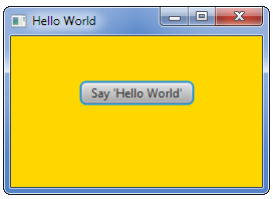
3. When the CSS file is created, add the following style definitions into it.

```
.root {
                    -fx-background-color: gold;
                    }
                    .label {
                    -fx-font-size: 20;>
                    }
```

   The first of the styles makes the background in the application window "gold" and the second one - sets the font size for the text `Hello World!` to 20 pixels.



4. Run the application again to see the result (`Shift+F10`).



Now that you've brought the application to a reasonable state, you may want to package it. For corresponding instructions, see Packaging JavaFX Applications .

Opening FXML files in JavaFX Scene Builder

When you open an FXML file in the editor, there are two tabs underneath the editing area:

- Text. This tab is for developing the markup.
- Scene Builder. This tab is for editing the file in JavaFX Scene Builder . (You should be using the Scene Builder version 2.x.)

  If you haven't specified the path to Scene Builder yet, there is the text *Please configure JavaFX Scene Builder path*.

  Click the path link and select the Scene Builder executable file in the dialog that opens.

  After a while, the FXML file will open in Scence Builder.

If you are using Scene Builder version 1.x, the Scene Builder tab is empty and you should use the following procedure (this works for version 2.x too):

1. Select the FXML file of interest in the Project tool window, or open the file in the editor.
2. From the context menu, select Open In SceneBuilder. (If you are in the editor, this applies to the Text tab.)

Packaging JavaFX Applications

You can package your JavaFX application by building the corresponding artifact . For JavaFX applications, IntelliJ IDEA provides a dedicated artifact type (JavaFx Application). One JavaFx Application artifact configuration is created by IntelliJ IDEA automatically if you create a project as described in To create a project for JavaFX application development from scratch.

Alternatively, you can generate an Ant build file, and then use that build file to package your application.

- Building an artifact
- Generating and using an Ant build file

See also, Applications with a Preloader: Project Organization and Packaging .

Building an artifact

To build an artifact for your JavaFX application:

- Select Build | Build Artifacts. Then, in the Build Artifacts | Action popup, select the artifact and select Build.
- Alternatively, you can turn on the Build on make option in the artifact configuration. In that case, an artifact will be built when making the project (e.g. Build | Make Project or `Ctrl+F9`).

By default, the artifact is generated in `<project_folder>\out\artifacts\<artifact_name>`.

See also, Working with Artifacts , Artifacts and Java FX Tab .

Generating and using an Ant build file

1. Generate an Ant build file for your project (Build | Generate Ant Build). For more information, see Generating Ant Build File .
2. Make the necessary changes to the generated build file.
3. Open the Ant Build tool window (e.g. View | Tool Windows | Ant Build), specify the build file to be used and run the necessary target. For more information, see Executing Ant Target .

See also, Ant .

Applications with a Preloader: Project Organization and Packaging

If you intend to package your application with a preloader , you, generally, should:

1. Create a project with at least two Java modules : one module for the application itself and one module for the preloader. For additional instructions, see Creating Projects and Adding Modules to a Project . See also, To create a project for JavaFX application development from scratch.
2. Develop the code for the application and the preloader.
3. Create two artifact configurations .
   - One of the configurations should be of the JavaFx Preloader type. This configuration will be used to build the preloader application (normally, a JAR file). The compilation output of the preloader module should be included in this artifact.
   - The second of the configurations - the one intended for packaging the application - should be of the JavaFx Application type. This artifact should include the compilation output of the application module and a copy of the preloader artifact.
     The copy of the preloader artifact should be added to the output root of the application artifact. (1. On the Output Layout tab for the application artifact, right-click <output root> and select Add Copy of | Artifact. 2. Select the preloader artifact in the Choose Artifacts dialog.)

   See Artifacts , Output Layout Tab and Java FX Tab .
4. Build the application artifact. As a result, a package containing both the application and the preloader will be generated.
   For additional instructions, see Building an artifact .

J2ME

IntelliJ IDEA supports the following toolkits for developing Java mobile applications:

– J2ME Wireless Toolkit (a.k.a. WTK), versions 1.x and 2.x.

– DoJa , version 1.x.

IntelliJ IDEA integration with those toolkits is based on the J2ME plugin . This plugin is bundled with the IDE and enabled by default.

Preparing for Java mobile application development

1. Download and install a JDK and one of the supported Java mobile toolkits .
2. Make sure that the J2ME plugin is enabled, see Making sure that the J2ME plugin is enabled.
3. Define the JDK and the mobile toolkit in IntelliJ IDEA. (In IntelliJ IDEA, the supported Java mobile toolkits are referred to as mobile SDKs). See Defining a JDK and a mobile SDK in IntelliJ IDEA .
4. Create a project with a J2ME module, see Creating a project with a J2ME module .
5. If necessary, configure the compilation settings specific to Java mobile applications, see Configuring Java mobile-specific compilation settings .

See also, Running and debugging Java mobile applications . For general instructions, see How-to .

Making sure that the J2ME plugin is enabled

Even though the J2ME plugin is enabled by default, it's always worth making sure that this plugin is still enabled before you start developing a Java mobile application. To do that:

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, under IDE Settings, select Plugins.
3. In the right-hand part of the dialog, on the Plugins page , type `j2` in the search box. As a result, only the plugins whose names and descriptions contain `j2` are shown in the list of plugins.
4. If the check box to the left of J2ME is not selected, select it.
5. Click OK in the Settings dialog.
6. If suggested, restart IntelliJ IDEA.

Defining a JDK and a mobile SDK in IntelliJ IDEA

To be able to develop Java mobile applications, you should define the JDK and the Java mobile toolkit that you are going to use in IntelliJ IDEA.

To define a JDK and a mobile SDK in IntelliJ IDEA

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the leftmost pane, under Platform Settings, click SDKs.
3. Above the pane to the right, click + and select JDK.
4. In the dialog that opens , select the installation directory of the JDK to be used and click OK.
5. Click + and select Mobile SDK.
6. In the dialog that opens , select the installation directory of your Java mobile toolkit (WTK or DoJa) and click OK.
7. If necessary, configure your custom profile and specify the preverify options on the SDK page (in the right-hand part of the dialog).
8. Click OK in the Project Structure dialog.

Creating a project with a J2ME module

To develop a Java mobile application, you need a project with a J2ME module .

To create a project with a J2ME module

1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen . Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.

2. On the first page of the wizard , in the left-hand pane, select J2ME.
   In the right-hand part of the page, specify the Java ME SDK to be used.

3. If necessary, enable SQL support and select the SQL dialect to be used by default.
   Click Next.

4. On the next page of the wizard , specify mobile SDK-specific options, and click Next.

5. Specify the name and location settings for your project and module. For more information, see Project Name and Location .
   Click Finish.

Configuring Java mobile-specific compilation settings

Generally, the compilation process for Java mobile applications includes the following stages:

- Compilation. During this stage a "standard" Java compiler such as javac is used. The related settings are specified as described in Configuring Compiler Settings .
- Preverification. Used at this stage is a preverify utility with an emulator included in the corresponding Java mobile SDK (WTK or DoJa). The related settings are specified as described later on this page.
- Packaging.

To configure the preverify settings

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. Select SDKs and then select your mobile SDK.
3. If you are using WTK, you can specify whether you want to use the default profile (MIDP) and configuration (CLDC) versions for a given emulator, or the custom profile and/or configuration versions. In the second case, list the profile and/or configuration versions in the corresponding fields.
4. For both WTK and DoJa, you can also specify the parameters for the preverify utility. For more information, see SDKs. Mobile .

Running and debugging Java mobile applications

To be able to run or debug your Java mobile application you should first create the corresponding run/debug configuration. Then, you should start this run/debug configuration in the run or the debug mode.

- Creating a J2ME run/debug configuration
- Starting a J2ME run/debug configuration

To create a J2ME run/debug configuration

1. Select Run | Edit Configurations.
2. In the Run/Debug Configurations dialog , click + and select J2ME.
3. Specify the configuration settings as necessary and click OK. For more information, see Run/Debug Configuration: J2ME .

To start your J2ME run/debug configuration

1. Make sure that the run/debug configuration for your mobile application is selected in the corresponding selector on the toolbar.
2. Now, to start this configuration in the run or the debug mode:
   - Select Run | Run or press `Shift+F10`.
   - Select Run | Debug or press `Shift+F9`.

JavaScript-Specific Guidelines

IntelliJ IDEA enables creating rich Internet applications and Web applications by providing elaborate support of JavaScript and tight integration with AJAX and other adjacent frameworks and technologies.

In this section:

- JavaScript-Specific Guidelines
  - Prerequisites
  - JavaScript support
  - Developing an application that contains JavaScript
- Configuring JavaScript Libraries
- Creating JSDoc Comments
- Viewing JavaScript Reference
- JavaScript-Specific Refactorings
- Configuring JavaScript Debugger
- Debugging JavaScript
- Unit Testing JavaScript
- Minifying JavaScript
- Tracing with Spy-Js
- Using AngularJS
- Using Angular CLI
- Using Bower Package Manager
- Using the Flow Type Checker
- Using Gulp Task Runner
- Using Grunt Task Runner
- Using JavaScript Code Quality Tools
- Using Meteor
- Using PhoneGap/Cordova
- Using ReactJS in JavaScript and TypeScript

Prerequisites

Before you start working with JavaScript, make sure that the JavaScript Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

JavaScript support

JavaScript files are marked with icon.

JavaScript support in IntelliJ IDEA includes:

- Switching between JavaScript language versions to choose the one that suits the targeted browser:
  - JavaScript 1.8.5
  - ECMAScript 3
  - ECMAScript 5.1
  - ECMAScript 6
  - JSX Harmony
  - Flow
- Full coding assistance:
  - Smart, DOM-based, browser-type aware JavaScript code completion for:
    - Keywords, labels, variables, parameters and functions.
    - User defined and built-in JavaScript functions.
    - JavaScript namespaces.
  - Error and syntax highlighting.
  - Code formatting and folding .
  - Numerous code inspections and quick-fixes , including the Switch to JavaScript <version> quick-fix.
  - Support of the strict mode standard.
  - Support for ECMAScript .
  - Initial support of ECMAScript Harmony .
- Code Generation
  - Generating code stubs based on file templates during file creation.
  - Inserting, expanding, and generating JavaScript code blocks using live templates .
  - Creating various applications elements via JavaScript and AJAX intention actions .
  - Possibility to create line and block comments (`Ctrl+Slash` / `Ctrl+Shift+Slash`).
  - Unwrapping and removing statements .
- Possibility to build and view type , method and call hierarchies
- Refactoring

  IntelliJ IDEA provides both common refactoring types available for all the supported languages and JavaScript-specific refactoring.
  - Common refactorings :
    - Rename a file, function, variable, parameter, or label (both directly and via references).

- Move/Copy a file.
- Safe Delete a file.
- Extract inlined script from HTML to a JS file.
- Extract function.

- JavaScript-Specific Extract Parameter Refactoring

- Numerous ways to navigate through the source code, among them:
  - Navigating with Structure Views .
  - Show/Goto Implementation (`Ctrl+Alt+B`) from overridden method / subclassed class.
- Advanced facilities for searching through the source code .
- Support of the JSDoc format and generating documentation comments .
- Viewing reference information:
  - Definitions, inline documentation, parameter hints.
  - JSDoc comments .
  - Lookup in external JavaScript libraries .
- Running and debugging.
  - Launching applications directly from IntelliJ IDEA by opening the starting application HTML page in the default IntelliJ IDEA browser .
  - A dedicated debug configuration for launching debugging sessions directly from IntelliJ IDEA.
  - A JavaScript-aware debugger that lets you execute applications step by step, evaluate expressions, examine related information and find runtime bugs.
  - Support for JavaScript breakpoints .
- Tight integration with related frameworks and technologies: AJAX , JQuery , YUI , DoJo , Prototype , MooTools , Qooxdoo , and Bindows :
  - Code completion for every framework.
  - DoJo style type annotations to provide more accurate completion and parameter type information.
  - Quick Documentation lookup for DoJo style commands.
- Support for the JSON (JavaScript Object Notation) format:
  - A JSON file type template mapped to `json` file extension.
  - JSON code formatting and folding .

To develop an application that contains JavaScript

Developing an application that contains JavaScript, generally, includes performing the following steps:

1. Make sure the JavaScript Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If it is not, enable the plugin . Make sure the JavaScript Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If it is not, enable the plugin . Make sure the JavaScript Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .
2. Create a project to implement your application. On the first page of the New Project wizard , select Static Web.
3. On the JavaScript page of the Settings dialog box, choose the JavaScript language version that suits the targeted browser.
4. Download, install, and configure JavaScript frameworks and libraries .
5. Populate the project . Use the following IntelliJ IDEA facilities, where applicable:
   - Coding assistance .
   - Code Generation .
   - Various types of navigation and search through the source code.
   - Viewing reference and generating documentation comments .
   - Look-up in external libraries .
6. Improve the quality and maintainability of your code using various types of refactoring , both common and JavaScript-specific.
7. Run your application by opening its starting HTML page in the IntelliJ IDEA default browser .
8. Debug your application.
   The JavaScript debugging functionality is incorporated in IntelliJ IDEA, so just configure the debugger , whereupon you can start the debugging session and proceed as usual : set the breakpoints, step through them, stop and resume the program, and examine it when suspended.

   Debugging for JavaScript applications is supported only in the Firefox and Google Chrome browsers.

Configuring JavaScript Libraries

On this page:

- Basics
  - Predefined and Custom Libraries
  - Visibility and Scope
- Viewing the Libraries Associated with a File
- Downloading and Installing a JavaScript-Related Library from IntelliJ IDEA
- Configuring a Custom JavaScript Library
  - Removing a Library File
  - Updating the Contents of a Library
  - Deleting a Library
  - Specifying the Scope to Use a Library in

Basics

When working in IntelliJ IDEA, you can use various JavaScript libraries and frameworks with full coding assistance at disposal. The only thing you need is "introduce" the required libraries to IntelliJ IDEA so they are recognized and references to them are successfully resolved.

Predefined and Custom Libraries

Right from IntelliJ IDEA, you can download and install a number of the most popular JavaScript libraries, such as: Dojo , ExtJS , jQuery , jQuery UI , Prototype , and others. Once installed, these libraries, by default, are visible in all files within the project.

You can also download any other JavaScript libraries and frameworks or even develop JavaScript libraries yourself, if necessary. To use such downloaded or self-developed libraries, you need to set them up as custom IntelliJ IDEA JavaScript libraries . Configured libraries can be used in code completion , highlighting, navigation , and Documentation Lookup .

To enable Documentation Lookup for symbols defined in an external library or framework, provide a link to its documentation. Upon pressing `Shift+F1` with the cursor positioned at the symbol in question, IntelliJ IDEA invokes this link and opens the documentation page in browser.

For jQuery , jQuery UI , Ext JS , Prototype , and Dojo library files, IntelliJ IDEA automatically detects links to the library documentation.

To get all the above mentioned coding assistance for a library or a framework, you need to configure it as a IntelliJ IDEA library and specify the library scope by associating the library with your project or its part, so you can reference the library from the files within this scope, get code completion, and retrieve definitions and documentation.

Please note, that configuring a framework as a IntelliJ IDEA library and associating it with a project only ensures coding assistance in the development environment, that is, while you are working in IntelliJ IDEA. To use a framework or library in the production environment, make sure the relevant version of the framework is available on the server.

Visibility and Scope

Each IntelliJ IDEA library is characterized by its visibility status and scope.

The scope of a library defines the set of files and folders in which the library is considered as library, that is, it is write-protected, excluded from check for errors and refactoring, but only affects the completion list and highlighting.

The visibility status of a library determines whether it can be used in one project (Project) or can be re-used at the IDE level (Global).

- Once configured, a Global library can be associated with any of IntelliJ IDEA projects. The library itself can be located wherever you need, its settings are stored with other IntelliJ IDEA settings in the dedicated directories under the IntelliJ IDEA home directory.
  The advantage of configuring a framework as a global library is that you can store such library in one location and re-use it in unlimited number of your projects without copying the library under the project root every time.

  The disadvantage of this approach is that to enable team work on a project all the team members should have the library stored on their machines in the same location relative to the project root.
- A Project library is visible only within one single project. Therefore a project library can be associated only with this project or its part. This means that project libraries cannot be re-used, so if you later try to use a framework configured as a project library with another project, you will have to configure the library anew.
  The advantage of configuring a JavaScript framework as a project library is that you can share the library definition among other team members through the project settings so each developer does not have to configure the library separately,
- Predefined libraries bring JavaScript definitions (also known as “stubs”) for the standard DOM, HTML5 and EcmaScript API, as well as for Node.js global objects. These libraries make the basis for coding assistance in accordance with the API provided by the corresponding JavaScript engine. By enabling a certain predefined library you can ensure that your code fits the target environment.
  Predefined libraries are by default enabled in the scope of the entire project. A predefined library can be disabled or associated with another scope but it cannot be deleted .

Viewing the Libraries Associated with a File

1. Open the file in the editor.
2. Click the Hector icon on the Status bar. The pop-up window lists the libraries associates with the current file. To change the list, click the Libraries in scope links and edit the scope settings in the Manage Scope dialog box that opens.

Downloading and Installing a JavaScript-Related Library from IntelliJ IDEA

IntelliJ IDEA provides a dedicated user interface for downloading and installing the most popular official JavaScript libraries. Using this interface, you can download and install Dojo , ExtJS , jQuery , jQuery UI , Prototype , and other libraries.

Besides the above listed official libraries, you can download stubs for TypeScript definition files .

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.

2. The JavaScript. Libraries page that opens shows a list of all the already available libraries. Click Download.
3. In the Download Library dialog that opens, choose the group of libraries in the drop-down list. The available options are Official libraries and TypeScript community stubs. Depending on your choice, IntelliJ IDEA displays a list of available libraries. Select the one to be downloaded and installed, and click Download and Install. You return to the JavaScript Libraries page where the new library is added to the list. CLick OK to save the settings.

Configuring a Custom JavaScript Library

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The JavaScript. Libraries page that opens shows a list of all the already available libraries. Click Add.
3. In the New Library dialog box that opens, specify the name of the library, the framework to configure the library from, and framework version to use.
4. Specify the library visibility :
   - To enable associating the library with the current project only, choose Current project.
   - To make the library available in any IntelliJ IDEA project, choose Global.
5. Create a list of files to be included in the library:
   1. Click the Add button + next to the list of library files and choose Attach Files or Attach Directory on the context menu, depending of whether you need separate files or an entire folder.
   2. Select the required file, files, or entire directory in the dialog box that opens. IntelliJ IDEA returns to the New Library dialog box where the Name read-only field shows the name of the selected library file or the names of relevant library files from the selected directory.
   3. In the Type field, specify which version of library you have downloaded and are going to add.
      - Choose Debug if you are adding a library file with uncompressed code. This version is helpful in the development environment, especially for debugging.
      - Choose Release if you are adding a library file with minified code. This version is helpful in the production environment because the file size is significantly smaller.

      It is recommended that you always have a debug version on hand along with the minified one. Minified code is hard to read and hard for IntelliJ IDEA to handle. When a debug version is available, IntelliJ IDEA automatically detects and ignores the minified file and retrieves definitions and documentation from the debug version.
   4. Specify the URL addresses to access the documentation for library files.
      - To add a link to the documentation for a library, select the corresponding library file, click the Specify button in the Documentation URLs area, and specify the documentation URL in the dialog box that opens.

        Tip For jQuery , jQuery UI , Ext JS , Prototype , and Dojo library files, IntelliJ IDEA automatically detects the link to the library documentation and suggests it in the Enter documentation URL text box.
      - To remove a link, select it in the Documentation URLs and click the Remove button.

Removing a Library File

In the New Library / Edit Library dialog box, select the required library file and click the Remove button − .

Updating the Contents of a Library

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The JavaScript. Libraries page that opens shows a list of all the already available libraries. Select the required library and click Edit.
3. In the Edit Library dialog box that opens, add library files, remove library files, and change links to documentation as necessary.

Deleting a Library

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The JavaScript. Libraries page that opens shows a list of all the already available libraries. Select the required library and click Remove.

Specifying the Scope to Use a Library in

The scope of a library defines the set of files and folders in which the library is considered as library, that is, it is write-protected, excluded from check for errors and refactoring, but only affects the completion list and highlighting.

By default, all predefined libraries and libraries downloaded from IntelliJ IDEA provide completion, resolution, highlighting and are treated as libraries in any file within the project. In other words, their usage scope is the whole project.

Libraries that you create yourself are not considered libraries in any of the files unless you specify their usage scope explicitly.

To specify the scope for a library:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The JavaScript. Libraries page that opens shows a list of all the already available libraries. Click Manage Scopes.
3. In the JavaScript Libraries. Usage Scope dialog box that opens, specify the custom JavaScript libraries to use in files and folders within your project. To appoint a library for a file or folder, select the required item in the File/Directory field and choose the relevant library from the Library drop-down list.
   The contents of the list depend on the visibility type of the configured libraries. Global libraries are on the list in all IntelliJ IDEA projects. Project libraries are on the list only within the project they were originally configured in.

Creating JSDoc Comments

To make your code easy to use by other developers it is considered good practice to provide an HTML documentation of its Application Programming Interface (API). Such documentation can be generated automatically by the JSDoc tool. All you need, is supply your code with documentation comments in accordance with the JSDoc standard. The tool retrieves information from your comments and renders it in HTML using a built-in template.

You can find a detailed description of the JSDoc syntax with examples and explanation of their use in the article An Introduction to JSDoc .

IntelliJ IDEA creates stubs of JSDoc comments on typing the opening tag `/**` and pressing `Enter`. If this feature is applied to a method or a function , `@param` tags are created. In any other places IntelliJ IDEA adds an empty documentation stub.

TODO patterns and Closure Compiler annotations inside documentation comments are also recognized and are involved in code completion, intention actions, and other types of coding assistance.

Documentation comments in your source code are available for the Quick Documentation Lookup and open for review on pressing `Ctrl+Q`.

IntelliJ IDEA checks syntax in the comments and treats it according to the Code Inspections settings.

On this page:

– Example of JavaScript comment
– Enabling generation of documentation comments
– Creating a JSDoc comment block

Example of JavaScript comment

Consider the following function:

```
function loadDocs(myParam1, myParam2){}
```

Type the opening documentation comment and press `Enter` to generate the documentation comment stub:

```
/**
* @param myParam1
* @param myParam2
*/
```

To enable or disable generation of documentation comments

1. Open the Settings dialog box , expand the Editor node, then expand General node, and click the Smart Keys page.
2. In the Enter section, select or clear Insert documentation comment stub check box.

To create a JSDoc comment block for a method or a function

1. Place the caret before the method or function declaration.
2. Type the opening block comment `/**` and press `Enter`.
3. Describe the listed parameters and return values.

Documentation comment can be created with the dedicated action Fix Doc Comment. It can be invoked by means of Find Action command.

Press `Ctrl+Shift+A`, with the caret somewhere within a class, method, function, or field, which should be documented, and enter the action name Fix Doc String. The missing documentation stub with the corresponding tags is added. For example:

```
/**
*
* @param userInput
* @return
*/
static boolean processRepeatConversion (@NotNull String userInput) {
boolean repeatConversion = false;
if (((userInput.equals("y")) || (userInput.equals("Y")))) {
repeatConversion = true;
}
return repeatConversion;

}
```

The next case lays with fixing problems in the existing documentation comments.

For example, if a method signature has been changed, IntelliJ IDEA highlights a tag that doesn't match the method signature, and suggests a quick fix.

For JavaScript, IntelliJ IDEA suggests an intention action UpdateJSDoc comment. You can also press `Ctrl+Shift+A`, and type the action name:

Tip The action Fix doc comment has no keyboard shortcut bound with it. You can configure keyboard shortcut of your own.

Viewing JavaScript Reference

Besides common referential procedures that are available in the context of any supported language, IntelliJ IDEA provides the following ways of retrieving JavaScript-specific reference:

- Viewing Inline Documentation
- Documentation Look-up in External JavaScript Libraries

Viewing Inline Documentation

Quick Documentation Lookup helps you get quick information for any symbol, provided that this symbol has been supplied with Documentation comments in the applicable format. IntelliJ IDEA recognizes inline documentation created in accordance with the JavaScript Documentation Tool format.

On this page:

– To view documentation for a symbol at caret, do one of the following
– To change the font size of quick documentation, do one of the following
– Quick Documentation Lookup Window
– Toolbar of the quick documentation lookup

To view documentation for a symbol at caret, do one of the following

– On the main menu, choose View | Quick Documentation Lookup.

– Press `Ctrl+Q`.

– Provided that the check box Show quick doc on mouse move in the editor settings is selected, just move your mouse pointer over the desired symbol.

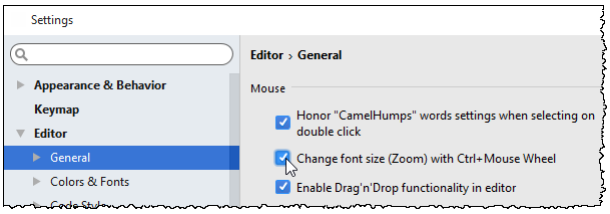
When you explicitly invoke code completion, then quick documentation for an entry selected in the suggestion list can be displayed automatically. The behavior of quick documentation lookup is configured in the Code Completion page of the Settings/Preferences dialog.

Tip Sequential pressing `Ctrl+Q` toggles the focus of the Quick Documentation pop-up window and Documentation Tool Window.

To change the font size of quick documentation, do one of the following

– Click in the upper-right corner of the quick documentation window, and move the slider.

– Rotate the mouse wheel while keeping the `Ctrl` key pressed.
Note that for this feature to work, you have to enable it in the General page of the editor settings.



Refer to the section Zooming in the Editor for details.

Quick Documentation Lookup Window

Toolbar of the quick documentation lookup

The Quick Documentation Lookup window helps navigate to the related symbols via hyperlinks, and provides a toolbar for moving back and forth through the already navigated pages, change font size, and viewing documentation in an external browser.

When pinned, the Quick Documentation Lookup turns into Documentation Tool Window, with the corresponding sidebar icon, and more controls.

Icon	Shortcut	Description
	`Left` or `Right`	Switch to the previous or next documentation page (e.g. after using hyperlinks). Note On an OS X computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.
	`Shift+F1`	View external documentation in the default browser.
	`F4`	Switch to the item (e.g. source) that corresponds to the documentation page currently shown.
		Turn the Auto-update from source option on or off. When the option is on, the information in the tool window is synchronized with your navigation in the editor and other places in the UI.
		Click this icon to show font size slider. Move the slider to increase or decrease the font size in the quick documentation window as required.

Documentation Look-up in External JavaScript Libraries

Besides common referential procedures and viewing JSDoc comments , you can also retrieve documentation for symbols defined in external JavaScript libraries and frameworks.

When this functionality is enabled, upon pressing `Shift+F1` with the cursor positioned at the symbol in question, IntelliJ IDEA invokes the corresponding link and opens the documentation page in the default IntelliJ IDEA browser .

- The default IntelliJ IDEA browser is configured on the Web Browsers page of the Settings dialog.
- For more details on specifying the browser to use by default, see Configuring Browsers .

To enable the external JavaScript library lookup functionality

1. Download the required libraries or frameworks.
2. Configure the downloads as libraries at the IntelliJ IDEA level.
3. Specify links to external documentation .

To view documentation on a symbol defined in an external JavaScript library

- Position the cursor at the symbol in question and choose View | External Documentation, or press `Shift+F1`.

JavaScript-Specific Refactorings

In addition to the full range of common refactorings , IntelliJ IDEA provides the following JavaScript-specific refactorings:

- Change Signature in JavaScript
- Extract Parameter in JavaScript
- Extract Variable in JavaScript

Change Signature in JavaScript

In JavaScript, you can use the Change Signature refactoring to:

- Change the function name.
- Add new parameters and remove the existing ones. Note that you can also add a parameter using a dedicated Extract Parameter refactoring.
- Reorder parameters.
- Change parameter names.
- Propagate new parameters through the method call hierarchy.

On this page:

- Examples
- Changing a function signature

Examples

The following table shows 4 different ways of performing the same Change Signature refactoring.

In all the cases, the function `result()` is renamed to `generate_result()` and a new parameter `input` is added to this function.

The examples show how the function call, the calling function (`show_result()`) and other code fragments may be affected depending on the refactoring settings.

Before After

```
// This is the function whose signature will be changed:

function$#160;result() {
// some code here
}

function show_result() {

// Here is the function call:

alert('Result:
' + result());
}

// Now we'll rename the function and
// add one (required) parameter.
```

```
// The function has been renamed to generate_result.
// The new parameter input has been added.

function generate_result(input) {
// some code here
}

function show_result() {

// The function call changed accordingly:

alert('Result:
' + generate_result(100));
}

// When performing the refactoring, 100 was specified as
// the parameter value.
```

```
// This is the function whose signature will be changed:

function
result() {

// some code here
}

function show_result() {

// Here is the function call:

alert('Result:
' + result());
}

// Now we'll rename the function and add one parameter.
// This time, we'll specify that the parameter is optional.
```

```
// The function has been renamed to generate_result.
// The new optional parameter input has been added.

function
generate_result(input) {
input = input || 100;
// some code here
}

function show_result() {

// The function call changed accordingly:

alert('Result:
' + generate_result(100));
}

// When performing the refactoring, 100 was specified as
// the parameter value.
```

```
// The function has been renamed to generate_result.
// The new parameter input has been added.

function
generate_result(input) {
// some code here
}

// Note the new function parameter:

function show_result(input) {

// The function call changed accordingly:

alert('Result:
' + generate_result(input));
}
```

```
// This is the function whose signature will be changed:
```

```
// This is the function whose signature will be changed:


                    function
                    result() {

                    // some code here
                    }

                    // This function will also change its signature:

                    function show_result() {


                    // Here is the function call:

                    alert('Result:
                    ' + result());
                    }

                    // Now we'll rename the function and add one optional
                    // parameter. We'll also ask IntelliJ IDEA to propagate
                    // the new parameter through the calling function show_result()
                    // to the function call.
```

```
// The function has been renamed to generate_result.
                    // The new optional parameter input has been added.

                    function
                    generate_result(input) {
                    input = input || 100;
                    // some code here
                    }

                    // Note the new function parameter:

                    function show_result(input) {
                    input = input || 100;

                    // The function call changed accordingly:

                    alert('Result:
                    ' + generate_result(input));
                    }

                    // When performing the refactoring, 100 was specified as
                    // the parameter value.
```

Changing a function signature

1. In the editor, place the cursor within the name of the function whose signature you want to change.

2. Do one of the following:
   - Press `Ctrl+F6`.
   - Choose Refactor | Change Signature in the main menu.
   - Select Refactor | Change Signature from the context menu.

3. In the Change Signature dialog , make the necessary changes to the function signature and specify which other, related changes are required.
   You can:
   - Change the function name. To do that, edit the text in the Name field.
   - Manage the function parameters using the table of parameters and the buttons to the right of it:
     - To add a new parameter, click (`Alt+Insert`) and specify the properties of the new parameter in the corresponding fields.
       When adding parameters, you may want to propagate these parameters to the functions that call the current function.
     - To remove a parameter, click any of the cells in the corresponding row and click (`Alt+Delete`).
     - To reorder the parameters, use (`Alt+Up`) and (`Alt+Down`). For example, if you wanted to make a certain parameter the first in the list, you would click any of the cells in the row corresponding to that parameter, and then click the required number of times.
     - To change the name of a parameter, make the necessary edits in the corresponding table cell.
   - Propagate new function parameters (if any) along the hierarchy of the functions that call the current function.
     (There may be the functions that call the function whose signature you are changing. These functions, in their turn, may be called by other functions, and so on. You can propagate the changes you are making to the parameters of the current function through the hierarchy of the calling functions and also specify which calling functions should be affected and which shouldn't.)

     To propagate the new parameters:
     1. Click (`Alt+G`).
     2. In the left-hand pane of the Select Methods to Propagate New Parameters dialog, expand the necessary nodes and select the check boxes next to the functions you want the new parameters to be propagated to.
        To help you select the necessary functions, the code for the calling function and the function being called is shown in the right-hand part of the dialog (in the Caller Method and Callee Method panes respectively).

        As you switch between the functions in the left-hand pane, the code in the right-hand part changes accordingly.
     3. Click OK.

4. To perform the refactoring right away, click Refactor.
   To see the expected changes and make the necessary adjustments prior to actually performing the refactoring, click Preview.

Extract Parameter in JavaScript

The Extract Parameter refactoring is used to add a new parameter to a method declaration and to update the method calls accordingly.

– Examples

– Extracting a Parameter in JavaScript

Examples

The following table shows two different ways of extracting a function parameter.

In the first of the examples a new parameter is extracted as an optional parameter. So the corresponding function call doesn't change.

In the second of the examples the parameter is extracted as a required parameter. So the corresponding function call changes accordingly.

Before **After**

```
// A new parameter will be added to this
                    // function to replace the 1's:

                    function calculate_sum(i) {

                    alert('Adding '
                    + 1 + ' to
                    ' + i);
                    return
                    1 + i;
                    }

                    function show_sum() {

                    // Here is the function call:

                    alert('Result:
                    ' + calculate_sum(5));
                    }

                    // When adding a new parameter we'll specify
                    // that it should be an optional one.
```

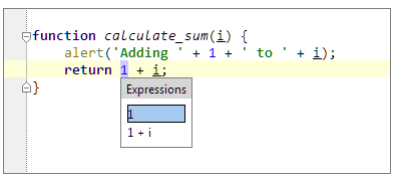
```
// The new parameter i2 has been added
                    // as an optional parameter:

                    function calculate_sum(i, i2) {
                    i2 = i2 || 1;
                    alert('Adding '
                    + i2 + ' to
                    ' + i);
                    return
                    i2 + i;
                    }

                    function show_sum() {

                    // The function call has not changed:

                    alert('Result:
                    ' + calculate_sum(5));
                    }
```

```
// A new parameter will be added to this
                    // function to replace the 1's:

                    function calculate_sum(i) {
                    alert('Adding '
                    + 1 + ' to
                    ' + i);
                    return
                    1 + i;
                    }

                    function show_sum() {

                    // Here is the function call:

                    alert('Result:
                    ' + calculate_sum(5));
                    }

                    // When adding a new parameter we'll specify
                    // that it should be a required one.
```

```
// The new parameter i2 has been added
                    // as a required parameter:

                    function calculate_sum(i, i2) {
                    alert('Adding '
                    + i2 + ' to
                    ' + i);
                    return i2 + i;
                    }

                    function show_sum() {

                    // The function call changed accordingly:

                    alert('Result:
                    ' + calculate_sum(5, 1));
                    }
```

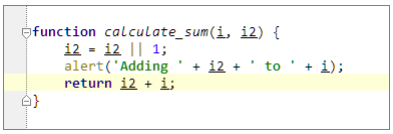
Extracting a Parameter in JavaScript

1. In the editor, place the cursor within the expression to be replaced by a parameter.
2. Do one of the following:
   – Press `Ctrl+Alt+P`.
   – Choose Refactor | Extract | Parameter in the main menu.
   – Choose Refactor | Extract | Parameter from the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



4. In the Extract Parameter dialog :
   1. Specify the parameter name in the Name field.

2. The Value field, initially, contains the expression that you have selected. Normally, you don't need to change this. (Depending on whether the option Optional parameter is selected or not, this will be the value assigned to the new parameter in the function body or passed to the function in the function calls.)
3. If, when extracting a new parameter, you don't want to change the function calls, select the Optional parameter check box. If you do so, the value specified in the Value field will be assigned to the new parameter in the function body. The calls to the function (if any) won't change.
   If you want to pass the value (specified in the Value field) to the new parameter through the existing function calls, clear the Optional parameter check box. If you do so, all the function calls will change according to the new function signature; no explicit assignment of the parameter value will be added to the function body.
4. If more than one occurrence of the expression is found within the function body, you can choose to replace only the selected occurrence or all the found occurrences with the references to the new parameter. Use the Replace all occurrences check box to specify your intention.
5. Click OK.



Extract Variable in JavaScript

In JavaScript and ActionScript (if supported by your version of IntelliJ IDEA), you can replace an expression with a variable or a constant. For JavaScript 1.7 or a later version, there is also an option of extracting a local variable.

To perform this refactoring, you can use:

- In-place refactoring . In this case you specify the new name right in the editor.
- Refactoring dialog , where you specify all the required information. To make such a dialog accessible, you have to clear the check box Enable in-place mode in the editor settings.

JavaScript Examples

Before After

```
Parenizor.method('toString', function ()
                    {
                    return '(' + this.getValue() + ')';
                    }
```

GLOBAL VARIABLE

```
Parenizor.method('toString', function ()
                    {
                    var string = '(' + this.getValue()
                    + ')';
                    return string;
                    }
```

LOCAL VARIABLE

```
Parenizor.method('toString', function ()
                    {
                    let string = '(' + this.getValue()
                    + ')';
                    return string;
                    }
```

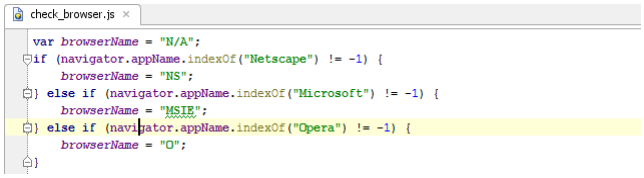
```
var browserName = "N/A";
                    if (navigator.appName.indexOf("Netscape") != -1) {
                    browserName = "NS";
                    } else if (navigator.appName.indexOf("Microsoft") != -1) {
                    browserName = "MSIE";
                    } else if (navigator.appName.indexOf("Opera") != -1) {
                    browserName = "O";
                    }
```

```
var browserName = "N/A";
                    var appName = navigator.appName;
                    if (appName.indexOf("Netscape") != -1) {
                    browserName = "NS";
                    } else if (appName.indexOf("Microsoft") != -1) {
                    browserName = "MSIE";
                    } else if (appName.indexOf("Opera")
                    != -1) {
                    browserName = "O";
                    }
```

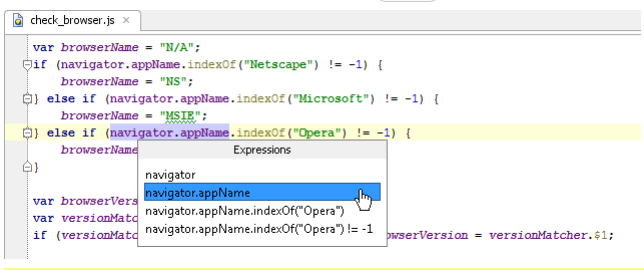
To extract a variable using in-place refactoring

1. In the editor, select the expression to be replaced with a variable. You can do that yourself or use the smart expression selection feature to let IntelliJ IDEA help you. So, do one of the following:
   - Highlight the expression. Then choose Refactor | Extract | Variable on the main menu or on the context menu.
     Alternatively, press `Ctrl+Alt+V` .

   - Place the cursor before or within the expression. Choose Refactor | Extract Variable on the main menu or on the context menu. or press `Ctrl+Alt+V` .



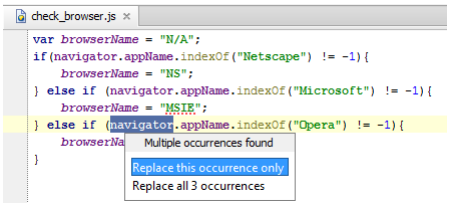
     In the Expressions pop-up menu, select the expression. To do that, click the required expression. Alternatively, use the Up and Down arrow keys to navigate to the expression of interest, and then press `Enter` to select it.



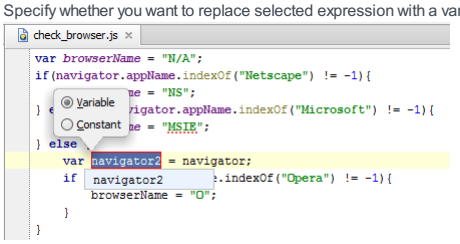
Note The Expressions pop-up menu contains all the expressions appropriate for the current cursor position in the editor.

When you navigate through the suggested expressions in the pop-up, the code highlighting in the editor changes accordingly.

2. If more than one occurrence of the selected expression is found, select Replace this occurrence only or Replace all occurrences in the Multiple occurrences found pop-up menu. To select the required option, just click it. Alternatively, use the `Up` and `Down` arrow keys to navigate to the option of interest, and press `Enter` to select it.



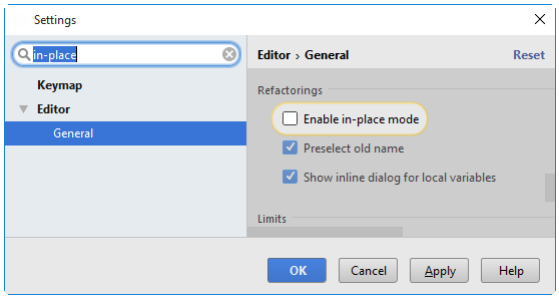
3. Specify whether you want to replace selected expression with a variable, or a constant. To do that, click the desired radio button in the pop-up balloon:



4. Specify the name of the variable. Do one of the following:
   - Select one of the suggested names from the pop-up list. To do that, double-click the suitable name. Alternatively, use the `Up` and `Down` arrow keys to navigate to the name of interest, and `Enter` to select it. When finished, press `Escape`.
   - Edit the name by typing. The name is shown in the box with red borders and changes as you type. When finished, press `Escape`.

To extract a variable using the dialog box

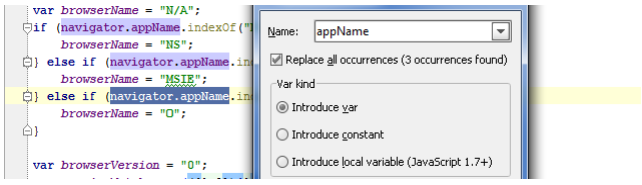
If the Enable in place refactorings check box is cleared in the Editor settings, the Extract Variable refactoring is performed by means of the dialog box.



1. Select the desired expression, and invoke Extract Variable refactoring as described above .

2. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.

3. In the Extract Variable dialog :
   - Specify the variable name next to Name. You can select one of the suggested names from the list or type the name in the Name box.
   - If more than one occurrence of the selected expression is found, you can select to replace all the found occurrences by selecting the corresponding check box. If you want to replace only the current occurrence, clear the Replace all occurrences check box.
   - Specify the type of variable to be extracted: a (global) variable, a constant, or a local variable. To do that, click the required option in the Var kind area. (Note that JavaScript is not supported in IntelliJ IDEA Community Edition.)
   - For ActionScript, you can choose to introduce a constant rather than a variable. To do that, select the Make constant check box. Note that ActionScript is not supported in IntelliJ IDEA Community Edition.)
   - Click OK.



Configuring JavaScript Debugger

Debugging JavaScript in IntelliJ IDEA is supported through the JavaScript Debugger. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Debugging of JavaScript code is supported in Google Chrome and other browsers of the Chrome by means of browser-specific JetBrains extensions. No special steps are required from your side to install or enable these extensions. IntelliJ IDEA does it for when you initiate a debugging session for the first time.

During a debugging session, IntelliJ IDEA considers any injected JavaScript code in a Web page opened in the browser as subject for debugging. This means that you cannot use the same instance of a browser for browsing and for debugging simultaneously. For Google Chrome you can solve this problem by configuring IntelliJ IDEA to use a separate Chrome user profile for debugging.

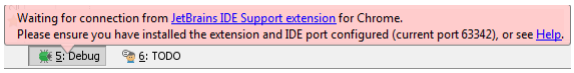
On this page:

– Common JavaScript debugger options

Configuring the Settings for Debugging JavaScript Code

1. Open the JavaScript page: choose File | Settings and in the Settings dialog box that opens, click Debugger under Build, Execution, Deployment.

2. On the JavaScript page that opens, specify the following:
   – Use the Built-in server port spin-box to specify the number of the port where the built-in web server runs. The port number should be set to 1024 or higher. By default this port is set to the default IntelliJ IDEA port through which IntelliJ IDEA accepts connections from services (`63342`). During a debugging session, the Chrome extension will listen to this port. If for some reason the default IntelliJ IDEA port is already busy, IntelliJ IDEA finds the closest available port and starts on it. This results in a conflict: IntelliJ IDEA is running on a "new" port while the Chrome extension still listens to the port of a previously started product and fails to establish connection with IntelliJ IDEA. The conflict reveals when you initiate a debugging session from IntelliJ IDEA: the extension fails to connect through the default port, IntelliJ IDEA waits for connection from the extension and displays the following message with the port number where it is actually running:

   To fix the problem, specify the actual port in the Chrome extension options, see Using JetBrains Chrome Extension .
   – In the Value tooltip delay (ms) field , specify the time period between pointing at an object and displaying the object's value in a tooltip.
   – Whether to display DOM properties in a tab or not.
   – Whether to display function values in a tab or not. If you want the function values to be shown only for user-defined functions, select the Show only user-defined functions. (Otherwise, the function values will be shown for all function.)
   – Whether you want object node properties to be shown. If so, specify the properties. Use + and — to manage the list of properties.
   – Whether you want the debugger to skip certain scripts. If so, specify URLs for the corresponding scripts. Use + and — to manage the list of script URLs.

Debugging JavaScript

On this page:

- Basic Terms
- Starting a Debugging Session by Opening an HTML File from IntelliJ IDEA (Local Debugging)
- Starting a Debugging Session through a User-defined Configuration (Local Debugging)
- Local Debugging in Firefox
- Remote Debugging
- Mapping
- Debugging Asynchronous Code
- Debugging Workers

Basic Terms

When an HTML file with injected JavaScript code is opened in the browser, the script execution starts and suspends on reaching the first breakpoint, whereupon you can analyze the suspended program, step through it, resume execution when ready, etc. This HTML file does not necessarily have to be the one that implements the starting page of the application.

Depending on the location of the application files, IntelliJ IDEA distinguishes between local and remote debugging modes.

1. Local debugging. In this mode, the application files remain where they are in the IntelliJ IDEA project on you machine. IntelliJ IDEA runs them on the built-in Web server with the root URL `http://localhost:<built-in server port>/<project root>`. The built-in server port (1024 or higher) is specified on the JavaScript page of the Settings dialog box. A debugging session in this mode can be initiated in two ways:
   - Open the HTML file with the JavaScript injection to be tested and launch the temporary run/debug configuration that IntelliJ IDEA has generated automatically. This approach it helpful when you do not need to debug the entire application but just one script.
   - Create and launch a permanent debug configuration.
2. Remote debugging. In this mode, the application files are on an external web server and you have their copies in a IntelliJ IDEA project on your computer. No matter, whether the web server itself is running on a physically remote host or on your machine, application files deployed on it are treated as remote. For details, see Deployment. Working with Web Servers .
   You can deploy the application files to a web server and launch the application yourself or debug an already running application.

   When a remote HTML file with a JavaScript injection is opened, the debugger tells IntelliJ IDEA the name of the currently processed file and the number of the line to be processed. IntelliJ IDEA opens the local copy of this file and indicates the line with the provided number. This behaviour is enabled by specifying correspondence between files and folders on the server and their local copies. This correspondence is called mapping, it is set in the debug configuration.

   A Remote debugging session can be initiated only through a permanent debug configuration that you have to define yourself manually.

IntelliJ IDEA supports local and remote JavaScript debugging only in Google Chrome and other browsers of the Chrome family. In Firefox , remote debugging is not supported at all. Local debugging is supported for Firefox, version 36 an higher, through the Firefox Remote debug configuration.

Starting a Debugging Session by Opening an HTML File from IntelliJ IDEA (Local Debugging)

1. Set the breakpoints in the JavaScript code, as required.
2. Open the HTML file with the JavaScript injection to debug or select the HTML file in the Project view .
3. On the context menu of the editor or the selection, choose Debug <file name>. The automatically generated debug configuration starts a debugging session. This configuration is temporary but you can save it later.
   The file opens in the default browser and the Debug tool window appears.
4. If you are using Chrome or Dartium, you need a Chrome extension. If the extension is not installed and activated in your browser, the Debug tool window shows a message with a link to the web store where the extension is available. Click the link and install the extension. For details, see Using JetBrains Chrome Extension .
5. In the Debug tool window, proceed as usual: step through the program , stop and resume program execution, examine it when suspended , view actual HTML DOM , edit the code , etc.

Starting a Debugging Session through a User-defined Configuration (Local Debugging)

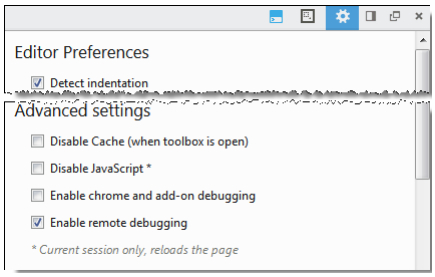
1. Set the breakpoints in the JavaScript code, as required.
2. Create a permanent debug configuration.
   1. On the main menu, choose Run | Edit Configurations.
   2. Click the Add New Configuration toolbar button +, and choose JavaScript Debug on the context menu.
   3. In the Run/Debug Configuration: JavaScript Debug dialog box that opens, specify the following:
      - The URL address of the HTML file that implements the page to start debugging from. Use the following format: `http://localhost:<built-in server port>/<project root>/<path to the HTML file relative to the project root>`
      - Choose the browser to debug the application in (Firefox or Chrome).
   4. Click OK to save the configuration settings.
3. Choose the newly created configuration in the Select run/debug configuration drop-down list on the tool bar and click the Debug toolbar button .
   The HTML file specified in the run configuration opens in the chosen browser and the Debug tool window appears.
4. If you are using Chrome or Dartium, you need a Chrome extension. If the extension is not installed and activated in your browser, the Debug tool window shows a message with a link to the web store where the extension is available. Click the link and install the extension. For details, see Using JetBrains Chrome Extension .
5. In the Debug tool window, proceed as usual: step through the program , stop and resume program execution, examine it when suspended , view actual HTML DOM , edit the code , etc.

Local Debugging in Firefox

Due to the recent crucial changes in the Firefox architecture, neither remote debugging nor standard for IntelliJ IDEA local debugging work for Firefox. The procedure below is applicable to Firefox versions 36 and higher.

1. Enable debugging in Firefox:
   1. Open your Firefox browser, then open Tools | Web Seveloper | Toggle Tools.

2. In the Development Tools pane that opens, click Toolbox Options button on the toolbar and select the Enable remote debugging check box under Advanced Settings.



3. In the console at the bottom of the browser (the console appeared opened when you opened the Developer toolbar), type `listen <port number>`.

```
» listen 6000
```

You can specify any port number, however it is recommended that you use 6000 and higher. Later you will specify this port number in the run configuration.

If you completed the above mentioned steps successfully, a pop-up window with the following message is shown: Listening on port 6000.

2. Create a run/debug configuration:
   1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
   2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button, and choose Firefox Remote on the context menu. Do not feel mislead by the term Remote. In this context it reflects the technical details of the implementation.
   3. In the Run/Debug Configuration: Firefox Remote dialog box that opens, specify the host where the application is running (currently it is always `localhost`) and the port the debugger will listen to. It must be the port that you specified when enabling debugging in Firefox. The default value is 6000.
   4. Click OK to save the configuration settings.
3. Set the breakpoints where necessary.
4. Open the HTML file with the related JavaScript code the editor or select it in the Project tool window, choose Open in Browser on the context menu of the selection, then choose Firefox. The browser opens showing the application after the code execution, that is, the breakpoints you set have no effect yet.
5. Choose the newly created configuration in the Select run/debug configuration drop-down list on the toolbar and click the Debug toolbar button. In the dialog bo that opens, click OK to allow incoming connections. From the Choose Page to Debug pop-up list, choose the page with your application.
6. Refresh the page with your application in Firefox: the page shows the results of code execution up to the first breakpoint.
7. In the Debug tool window, proceed as usual: step through the program , stop and resume program execution, examine it when suspended , view actual HTML DOM , edit the code , etc.

Remote Debugging

1. Set the breakpoints in the JavaScript code, as required.
2. Create a permanent debug configuration.
   1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
   2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button, and choose JavaScript Debug on the context menu.
   3. In the Run/Debug Configuration: JavaScript Debug dialog box that opens, specify the following:
      - The URL address of the HTML file that implements the page to start debugging from. This URL address should correspond with the server access configuration.
      - Choose the browser to debug the application in (Chrome or another browser of the Chrome family).
      - Optionally define mappings between local files and URL addresses of their copies on the server. These mappings are required only if the local folder structure under the project root differs from the folder structure on the server. If the structures are identical, IntelliJ IDEA itself "retrieves" the paths to local files by parsing the URL addresses of their copies on the server.

      Click OK to save the configuration settings.
3. Choose the newly created configuration in the Select run/debug configuration drop-down list on the tool bar and click the Debug toolbar button.
   The HTML file specified in the run configuration opens in the chosen browser and the Debug tool window appears.
4. If you are using Chrome or Dartium, you need a Chrome extension. If the extension is not installed and activated in your browser, the Debug tool window shows a message with a link to the web store where the extension is available. Click the link and install the extension. For details, see Using JetBrains Chrome Extension .
5. In the Debug tool window, proceed as usual: step through the program , stop and resume program execution, examine it when suspended , view actual HTML DOM , edit the code , etc.

Mapping

If you are using a third-party Web server (not the built-in IntelliJ IDEA server) for debugging or you have defined several resource root folders in addition to the project root, you need to map files, folders, or the entire project folder to the files or folders on the server. When a remote HTML file with a JavaScript injection is opened, the debugger tells IntelliJ IDEA the name of the currently processed file and the number of the line to be processed. IntelliJ IDEA opens the local copy of this file and indicates the line with the provided number. This behaviour is enabled by specifying correspondence between files and folders on the server and their local copies. This correspondence is called mapping, it is set in the debug configuration.

To configure mappings:

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button, and choose JavaScript Debug on the context menu.
3. In the Run/Debug Configuration: JavaScript Debug dialog box that opens, map the files or folders or the entire project folder to URL addresses of files and folders on the server.

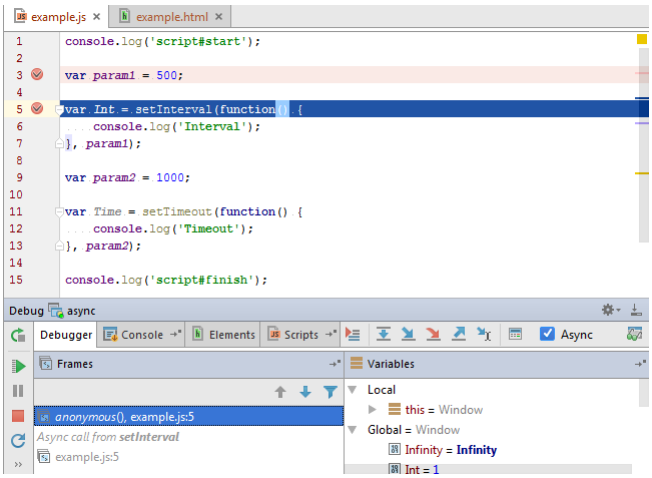
Debugging Asynchronous Code

IntelliJ IDEA supports debugging asynchronous client-side JavaScript code, currently this functionality is available only in Chrome. The asynchronous debugging mode is toggled through the Async check box on the tool bar of the Debugger tab in the `Debug` tool window. The check box is displayed only during a JavaScript debugging session.

When this check box is selected, IntelliJ IDEA recognizes breakpoints inside asynchronous code and stops at them and lets you step into asynchronous code. As soon as

- When this check box is selected, IntelliJ IDEA recognizes breakpoints inside asynchronous code and stops at them and lets you step into asynchronous code. As soon as a breakpoint inside an asynchronous function is hit or you step into asynchronous code, a new element `Async call from <caller>` is added in the Frames pane of the Debugger tab. IntelliJ IDEA displays a full call stack, including the caller and the entire way to the beginning of the asynchronous actions.
- When the check box is cleared, IntelliJ IDEA does not recognize and therefore skips breakpoints in the asynchronous code and does not allow you to step into it.

The image below shows an example of a JavaScript debugging session.



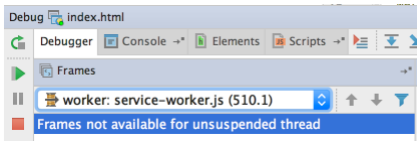
- With the Async check box selected, the debugger will stop at line3(breakpont), then at line5(breakpoint). On clicking Step into, the debugger will stop at line5 (on `function`), then will move to line6.
- With the Async check box cleared, the debugger will stop at line3(breakpont), then at line5(breakpoint). On clicking Step into, the debugger will move to line9.

Debugging Workers

IntelliJ IDEA supports debugging Service Workers and Web Workers. IntelliJ IDEA recognizes breakpoints in each worker and shows the debug data for it as a separate thread in the Frame pane on the Debugger tab of the Debug Tool Window.

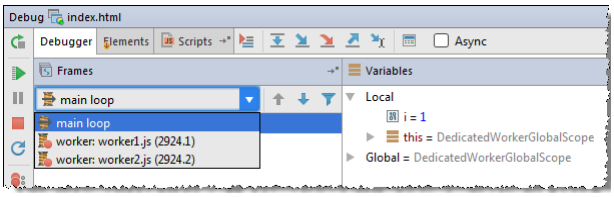
Note that IntelliJ IDEA can debug only dedicated workers, debugging for shared workers is currently no supported.

1. Set the breakpoints in the Workers to debug.
2. If you are using Service Workers, make sure the Allow unsigned requests check box on the Debugger page is selected. Otherwise your service workers may be unavailable during a debug session:



3. Create a JavaScript Debug run/debug configuration:
   1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
   2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button +, and choose JavaScript Debug on the context menu.
   3. In the Run/Debug Configuration: JavaScript Debug dialog box that opens, specify the following:
      - The URL address of the HTML file that implements the page to start debugging from. This URL address should correspond with the server access configuration.
      - Choose the browser to debug the application in (Chrome or another browser of the Chrome family).
      - Optionally define mappings between local files and URL addresses of their copies on the server. These mappings are required only if the local folder structure under the project root differs from the folder structure on the server. If the structures are identical, IntelliJ IDEA itself "retrieves" the paths to local files by parsing the URL addresses of their copies on the server.

      Click OK to save the configuration settings.
4. Choose the newly created configuration in the Select run/debug configuration drop-down list on the tool bar and click the Debug toolbar button.
   The HTML file specified in the run configuration opens in the chosen browser and the Debug tool window opens with the Frames drop-down list showing all the Workers:



To examine the data (variables, watches, etc.) for a Worker, select its thread in the list and view its data in the Variables and Debug Tool Window. Watches panes. When you select another Worker, the contents of the panes are updated accordingly.

Unit Testing JavaScript

IntelliJ IDEA provides facilities for running JavaScript unit tests against a local or remote test server. You can run test cases, methods, or entire test files.

To create and run unit tests in JavaScript, perform these general steps

1. Enable JavaScript unit testing support .
2. Write the unit tests .
3. Mark the folder where the unit tests are stored as a test source folder, see Configuring Content Roots .
4. If applicable, write configuration files that define which test and corresponding production files should be loaded into the browser and in which order.
5. Start the test server and capture a browser.
6. Launch unit tests and monitor test results in the Run tool window.

The topics in this part provide guidelines in JavaScript-specific unit testing procedures.

- Enabling JavaScript Unit Testing Support
- Creating JavaScript Unit Tests
- Running JavaScript Unit Tests
- Monitoring Code Coverage for JavaScript

For general information on testing in IntelliJ IDEA, see the section Testing .

Enabling JavaScript Unit Testing Support

With IntelliJ IDEA, you can run and debug unit tests using one of the following test runners:

- JSTestDriver
- Karma
- Cucumber.js

To write unit tests that are recognized and processed by these test runners, use the following JavaScript unit testing frameworks:

- JSTestDriver Assertion framework. The framework libraries are bundled with the JSTestDriver plugin.
- Jasmine .
- QUnit .
- Mocha

In this part:

- Preparing to Use JSTestDriver Test Runner
- Preparing to Use Karma Test Runner
- Preparing to Use Mocha Test Framework

Preparing to Use JSTestDriver Test Runner

On this page:

- Configuring the JSTestDriver test runner
- Configuring testing frameworks in a project

Configuring the JSTestDriver test runner

You do not need to download the JSTestDriver framework manually. The server and the assertion framework are provided through the JSTestDriver plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

This plugin does the following:

- Runs the JSTestDriver server that captures an opened browser to execute tests in.
- During the test creation, detects the unit testing framework the test code complies with, whereupon suggests the Add <test framework> support intention action , provided that the framework is recognized as a IntelliJ IDEA JavaScript library and is thus available in the IDE.

Configuring testing frameworks in a project

1. Download the framework of your choice and configure it as a IntelliJ IDEA JavaScript library .
2. Do one of the following:
   - Add the project folder to the library scope .
   - Enable the framework support on-the-fly during test creation . Write the test as required, position the cursor at it, and press `Alt+Enter` . Then choose the Add <test framework> support intention action from the list.
   - To use Jasmine, add `jasmine-jstd-adapter` to the configuration file.
     Open `jsTestDriver.conf` and type the following code in it:

```
load:

lib/jasmine/jasmine.js
lib/jasmine-jstd-adapter/JasmineAdapter.js
```

For more details about JSTestDriver configuration files, see Running JavaScript Unit Tests .

Preparing to Use Karma Test Runner

The Karma test runner supports executing unit tests against the Node.js server. The tests themselves can use Jasmine , QUnit , or Mocha libraries in Karma-specific edition.

Besides running unit tests, with Karma you can measure how much of your code is covered with tests. For more details, see Monitoring Code Coverage for JavaScript . To run tests with coverage, you need an additional Karma-related package.

The easiest way to install the Karma test runner is to use the Node Package Manager (npm), which is a part of Node.js . See Installing and Removing External Software Using Node Package Manager for details.

Depending on the desired location of the Karma test runner executable file, choose one of the following methods:

- Install the test runner globally at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project.
- Install the test runner in a specific project and thus restrict its use to this project.
- Install the test runner in a project as a development dependency .

In either installation mode, make sure that the parent folder of the Karma test runner is added to the `PATH` variable. This enables you to launch the test runner from any folder.

IntelliJ IDEA provides user interface both for global and project installation as well as supports installation through the command line.

On this page:

- Preparing to Install the Karma Test Runner
- Installing Karma Globally
- Installing Karma in a Project
- Configuring Testing Frameworks in a Project

Preparing to Install the Karma Test Runner

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The Karma test runner is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the Karma test runner.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Karma test runner and npm from any folder.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Install and enable the Karma repository plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Installing Karma Globally

Global installation makes a test runner available at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project. Moreover, during installation the parent folder of the test runner is automatically added to the `PATH` variable, which enables you to launch the test runner from any folder. To install the test runner globally, do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
  3. Type the following command at the command line prompt:

     ```
     npm install -g karma
     ```

     The `-g` key makes the test runner run in the global mode. Because the installation is performed through NPM, the Karma test runner is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the test runner from any folder.

     For more details on the NPM operation modes, see npm documentation . For more information about installing the Karma test runner, see https://npmjs.org/package/karma .
- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
  3. In the Available Packages dialog box that opens, select the required package to install.
  4. Select the Options check box and type `-g` in the text box next to it.
  5. Optionally specify the product version and click Install Package to start installation.

Installing Karma in a Project

Installing a test runner in a specific project restricts its use to this project. To run project installation, do one of the following:

- Run the installation from the command line:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install karma
```

– Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in template-based projects of the type Node Boilerplate or Node.js Express, which already have the `node_modules` folder. The latter is important because NPM installs the Karma test runner in a `node_modules` folder. If your project already contains such folder, the Karma test runner is installed there.

Projects of other types or empty projects may not have a `node_modules` folder. In this case npm goes upwards in the folder tree and installs the Karma test runner in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be outside your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the Karma test runner is installed there.

In either case, make sure that the parent folder of the Karma test runner is added to the `PATH` variable. This enables you to launch the test runner from any folder.

Configuring Testing Frameworks in a Project

1. Download the Jasmine, QUnit, or Mocha framework. The easiest way is to use the Node.js Package Manager:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
   2. On the Node.js and NPM page that opens, click Install in the Packages area.
   3. In the Available Packages dialog box that opens, select `karma-jasmine`, `karma-qunit`, or `karma-mocha` package and click Install Package. Close the dialog box when ready. IntelliJ IDEA returns to the Node.js page, where the selected package is added to the Packages list. Click OK.
2. To enable IntelliJ IDEA to resolve references to the downloaded framework and provide code completion and other types of coding assistance, configure the framework as a IntelliJ IDEA JavaScript library .

Preparing to Use Mocha Test Framework

The Mocha test framework supports executing unit tests against the Node.js server. The tests themselves can use BDD, TDD, Exports, and QUnit interfaces.

The easiest way to install the Mocha test framework is to use the Node Package Manager (npm), which is a part of Node.js. See Installing and Removing External Software Using Node Package Manager for details.

Depending on the desired location of the Mocha test framework executable file, choose one of the following methods:

- Install the test framework globally at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project.
- Install the test framework in a specific project and thus restrict its use to this project.
- Install the test framework in a project as a development dependency.

In either installation mode, make sure that the parent folder of the Mocha test framework is added to the `PATH` variable. This enables you to launch the test framework from any folder.

IntelliJ IDEA provides user interface both for global and project installation as well as supports installation through the command line.

On this page:

- Preparing to Install the Mocha Test Framework
- Installing Mocha Globally
- Installing Mocha in a Project

Preparing to Install the Mocha Test Framework

1. Download and install Node.js. The runtime environment is required for two reasons:
   - The Mocha test framework is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the Mocha test framework.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Mocha test framework and npm from any folder.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins.

Installing Mocha Globally

Global installation makes a test framework available at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project. Moreover, during installation the parent folder of the test framework is automatically added to the `PATH` variable, which enables you to launch the test framework from any folder. To install the test framework globally, do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs.
  3. Type the following command at the command line prompt:

     ```
     npm install -g mocha
     ```

     The `-g` key makes the test framework run in the global mode. Because the installation is performed through NPM, the Mocha test framework is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the test framework from any folder.

     For more details on the NPM operation modes, see npm documentation. For more information about installing the Mocha test framework, see https://npmjs.org/package/mocha.
- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
  3. In the Available Packages dialog box that opens, select the required package to install.
  4. Select the Options check box and type `-g` in the text box next to it.
  5. Optionally specify the product version and click Install Package to start installation.

Installing Mocha in a Project

Installing a test framework in a specific project restricts its use to this project. To run project installation, do one of the following:

- Run the installation from the command line:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the project root folder and type the following command at the command line prompt:

     ```
     npm install mocha
     ```
- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.

2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in template-based projects of the type Node Boilerplate or Node.js Express, which already have the `node_modules` folder. The latter is important because NPM installs the Mocha test framework in a `node_modules` folder. If your project already contains such folder, the Mocha test framework is installed there.

Projects of other types or empty projects may not have a `node_modules` folder. In this case npm goes upwards in the folder tree and installs the Mocha test framework in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be outside your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the Mocha test framework is installed there.

In either case, make sure that the parent folder of the Mocha test framework is added to the `PATH` variable. This enables you to launch the test framework from any folder.

Creating JavaScript Unit Tests

IntelliJ IDEA provides integration with JavaScript unit testing frameworks thus providing rich coding assistance for writing JavaScript unit tests. To get framework-aware coding assistance, download the desired framework, configure it as a IntelliJ IDEA JavaScript library, and define its visibility and scope.

The visibility status of a library determines whether it can be used in one project (Project) or can be re-used at the IDE level (Global).

The scope of a library defines the set of files and folders in which the library is considered as library, that is, it is write-protected, excluded from check for errors and refactoring, but only affects the completion list and highlighting.

For details, see Configuring JavaScript Libraries .

To create JavaScript unit tests, perform these general steps

- Create a folder `test` at the same level as the `src` folder.
- Populate the `test` folder. For each production file, create a separate test file and name it as follows: `<name of production file>.<Test>.js`.
- Mark the folder where the tests are stored as a test source folder, see Configuring Content Roots .
- If necessary, have IntelliJ IDEA detect and enable the required testing framework on-the-fly .

Running JavaScript Unit Tests

JavaScript unit tests are run in a browser against a test server which actually handles the testing process.

This server allows you to capture a browser to run tests in, loads the test targets to the captured browser, controls the testing process, and exchanges data between the browser and IntelliJ IDEA, so you can view test results without leaving the IDE.

The test server does not necessarily have to be on your machine, it can be launched right from IntelliJ IDEA through a test runner. Currently, IntelliJ IDEA supports integration with two test runners: JSTestDriver and Karma .

In this part:

- Running Unit Test on JSTestDriver
- Running Unit Tests on Karma
- Running Mocha Unit Tests

Running Unit Test on JSTestDriver

JavaScript unit tests are run in a browser against a test server which actually handles the testing process.

This server allows you to capture a browser to run tests in, loads the test targets to the captured browser, controls the testing process, and exchanges data between the browser and IntelliJ IDEA, so you can view test results without leaving the IDE.

The test server does not necessarily have to be on your machine, it can be launched right from IntelliJ IDEA through the JSTestDriver test runner.

The test server loads tests to the browser according to the configuration files that define which test and corresponding production files should be loaded and in which order. JSTestDriver server treats `*.jstd` and `*.conf` files as test runner configuration files. Create one or several configuration files in advance manually and then specify them in the run configuration .

On this page:

- Before You Start
- Creating a test runner configuration file manually
- Starting a IntelliJ IDEA default JSTestDriver test server
- Capture a browser to execute tests in
- Creating a JSTestDriver run configuration
- Launching unit tests

Before You Start

- Make sure the JSTestDriver repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Creating a test runner configuration file manually

A test runner configuration file defines which test and corresponding production files should be loaded and in which order. Any file with the extension `*.jstd` or `*.conf` is treated as a test runner configuration file.

1. In the Project tree, select the parent folder of the `src` (production) and the `test` folders, and choose New | File on the context menu.
2. In the New File dialog box, that opens, type a name of the configuration file with the extension `jstd` or `conf` .
3. Open the new file in the editor and specify the full path to the current folder and paths to the files to load relative to the current folder. Use wildcards to specify file name patterns. The required format is YAML , for more details, see description of test runner configuration files .

Starting a IntelliJ IDEA default JSTestDriver test server

If you are going to use a test server running on another machine or listening to another port, start it according to the server-specific instructions and specify its URL address in the Server ares of the Run/Debug Configuration: JSTestDriver dialog box.

1. Make sure the JSTestDriver plugin is enabled and at least one test runner configuration file is available in the project.
2. Open the JSTestDriver Server tool window by doing one of the following:
   - On the main menu, choose View | Tool Windows | JSTestDriver Server.
   - Click the Run button on the toolbar, and then click the Start a local server link in the pop-up window that opens.
3. In the JSTestDriver Server tool window, that opens, click the Run a local server toolbar button.

To stop the server when you are through with unit testing, click the Stop the local server toolbar button.

Capture a browser to execute tests in

1. Start the JSTestDriver Server if it is not running yet, and then switch to the JSTestDriver Server tool window.
2. To start a local browser with the Remote Console of the JSTestDriver, do one of the following:
   - Click the icon that indicates the browser of your choice.
   - If the browser is already opened, copy the contents of the Capture a browser using the URL read-only field and paste the URL in the address bar.

   In either case, the icon that indicates the chosen browser becomes active.

   You can have several browsers captured. However if you are going to debug your unit tests, you will have to appoint a browser for debugging in the Debug tab of the Run/Debug Configuration: JSTestDriver dialog box.
3. Switch to the JSTestDriver Server tool window and click the icon that indicates the browser you just opened. IntelliJ IDEA displays a message informing you that it is ready for executing tests.

Creating a JSTestDriver run configuration

1. Open the Run/Debug Configuration dialog box by doing one of the following:
   - On the main menu, choose Run | Edit Configurations.
   - Open the test file in the editor, and then choose Create <file name> on the context menu.
   - Select the test file in the Project tool window, and then choose Create <file name> on the context menu of the selection.
2. Click the Add button on the toolbar and select the JSTestDriver configuration type.
3. In the dialog box that opens, specify the test scope, configuration parameters, and activities to perform before test execution.
4. Apply the changes and close the dialog box.

Launching unit tests

1. To launch the tests according to a run configuration, select the JSTestDriver run/debug configuration from the list on the main toolbar. Then click the Run button to the right of the list.

2. To launch a single test, open the test file in the editor and click the blue arrow icon in the gutter area next to the text to run.

   Note The arrows appear only if the test framework used in your tests is associated with the project, so IntelliJ IDEA can recognize the tests, see Enabling JavaScript Unit Testing Support for details.

3. Monitor test execution in the Test Runner tab of the Run tool window as described in Monitoring and Managing Tests .

Running Unit Tests on Karma

JavaScript unit tests are run in a browser against a test server which actually handles the testing process.

This server allows you to capture a browser to run tests in, loads the test targets to the captured browser, controls the testing process, and exchanges data between the browser and IntelliJ IDEA, so you can view test results without leaving the IDE.

The test server does not necessarily have to be on your machine, it can be launched right from IntelliJ IDEA through the Karma test runner.

Karma executes unit tests according to a `karma.conf.js` configuration file which is generated semi-automatically in the interactive mode.

On this page:

– Before You Start
– Generating a Karma configuration file
– Creating a Karma run configuration
– Launching unit tests

Before You Start

1. Download and install Node.js . The runtime environment is required for two reasons:
   – The Karma test runner is started through Node.js.
   – NPM, which is a part of the runtime environment, is also the easiest way to download the Karma test runner.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Karma test runner and npm from any folder.
2. Make sure the Karma test runner is installed as described in Preparing to Use Karma Test Runner .
3. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
4. Install and enable the Karma repository plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Generating a Karma configuration file

Karma executes unit tests according to a `karma.conf.js` configuration file which is generated semi-automatically in the interactive mode. The instruction below ensures successful creation of a consistent configuration file `karma.conf.js` which in its turn ensures successful execution of the tests in your project. For more details, see http://karma-runner.github.io/0.10/config/configuration-file.html .

1. In the command line mode, switch to your project directory.
2. Type the following command at the command line prompt:

   ```
   karma init
   ```

   If Karma does not start, check the installation: the parent folder or the Karma executable file should be specified in the `PATH` variable.
3. Answer the questions to specify the following basic settings:
   – The testing framework to use.
   – The browsers to be captured automatically.
   – The patterns that define the location of test files to be involved in testing or excluded from it, for example, `src/*.js` and `test/*.js` . For more details, see http://karma-runner.github.io/0.10/config/files.html .

Creating a Karma run configuration

1. Open the Run/Debug Configuration dialog box by doing one of the following:
   – On the main menu, choose Run | Edit Configurations.
   – Open the test file in the editor, and then choose Create <file name> on the context menu.
   – Select the test file in the Project tool window, and then choose Create <file name> on the context menu of the selection.
2. Click the Add button + on the toolbar and select the Karma configuration type.
3. In the dialog box that opens, specify the location of the Node.js and Karma executable files and the path to the `karma.conf.js` configuration file.
4. Apply the changes and close the dialog box.

Launching unit tests

1. To launch the tests according to a run configuration, select the Karma run/debug configuration from the list on the main toolbar. Then click the Run button ▶ to the right of the list.
2. The Karma test server starts automatically without any steps from your side. View and analyze messages from the test server in the Karma Server tab of the Run tool window.
3. Monitor test execution in the Test Runner tab of the Run tool window as described in Monitoring and Managing Tests .

Running Mocha Unit Tests

On this page:

- Before You Start
- Creating a Mocha Run Configuration

Before You Start

1. Make sure the Node.js runtime environment and the Mocha framework are installed on your computer.
2. Make sure the NodeJS repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

See Preparing to Use Mocha Test Framework for details.

Creating a Mocha Run Configuration

1. Open the Run/Debug Configuration dialog box by doing one of the following:
   - On the main menu, choose Run | Edit Configurations.
   - Open the test file in the editor, and then choose Create <file name> on the context menu.
   - Select the test file in the Project tool window, and then choose Create <file name> on the context menu of the selection.

   Click the Add button + on the toolbar and select the Mocha configuration type. The Run/Debug Configuration: Mocha dialog box opens.
2. Specify the location of the Node.js executable file and the Mocha package.
3. Specify the working directory of the application. All references in test scripts will be resolved relative to this folder, unless such references use full paths.
   By default, the field shows the project root folder. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button ... and select the location in the dialog box, that opens.
4. Specify the tests to run:
   - All in directory: choose this option to run all the tests from files stored in a folder. In the Test directory field, specify the folder with the tests. To have IntelliJ IDEA look for tests in the subfolders under the specified directory, select the Include subdirectories check box.
   - File pattern: choose this option to have IntelliJ IDEA look for tests in all the files with the names that match a certain mask and specify the mask in the Test file pattern field.
   - Test file: choose this option to run only the tests from one file and specify the path to this file in the Test file field.
   - Suite: choose this option to run individual suites from a test file. In the Test file field, specify the file where the required suites are defined. Click the Suite name field and configure a list of suites to run. To add a suite to the list, click + and type the name of the required suite. To remove a suite, select it in the list and click –
   - Test: choose this option to run individual tests from a test file. In the Test file field, specify the file where the required tests are defined. Click the Test name field and configure a list of tests to run. To add a test to the list, click + and type the name of the required test. To remove a test, select it in the list and click –
5. Choose the interface used in the test to run.
6. Apply the changes and close the dialog box.

Monitoring Code Coverage for JavaScript

Code Coverage shows how much of your code is covered with tests and marks covered and uncovered lines visually right in the editor.

In this section:

– Prerequisites
– Enabling running tests with coverage in Karma
– Measuring code coverage for tests executed on JSTestDriver
– Measuring code coverage for tests executed on Karma

Prerequisites

1. The JSTestDriver or Karma test runner is installed and configured.
2. Additional third-party testing frameworks, for example, Jasmine , QUnit , or Mocha , are downloaded and integrated with IntelliJ IDEA .
3. A runner-specific configuration file is generated or written manually.

For more details, see Preparing to Use Karma Test Runner or Preparing to Use JSTestDriver Test Runner depending on your choice.

Enabling running tests with coverage in Karma

Coverage for tests run in Karma is calculated by the istanbul coverage tool. The tool is shipped within the `karma-coverage` package. Therefore running tests with coverage in Karma requires that you download the `karma-coverage` package and manually update the `karma.conf.js` configuration file.

The `karma-coverage` package should be installed in the same folder with Karma (`karma (library home)` package), no matter whether Karma is installed globally or in your project. Keep this restriction in mind when installing `karma-coverage` manually in the command line mode. If you are installing `karma-coverage` through the Node Package Manager interface, the proper installation folder will be chosen automatically.

1. Install the `karma-coverage` package through the Node Package Manager (npm).
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
   2. On the Node.js and NPM page that opens, click Install in the Packages area.
   3. In the Available Packages dialog box that opens, select the `karma-coverage` package, click Install Package, and close the dialog box when ready. IntelliJ IDEA returns to the Node.js page, where the `karma-coverage` package is added to the Packages list. Click OK.

   For details, see Installing and Removing External Software Using Node Package Manager .
2. Open the `karma.conf.js` configuration file and add the following information to it manually:
   1. Locate the `reporters` definition and add `coverage` to the list of values in the following format:

```
reporters: ['progress', 'coverage'],
```

   2. Add a `preprocessors` definition and specify the coverage scope according to the following format:

```
preprocessors: {
                    '**/*.js':
                    ['coverage']
                    }
```

Measuring code coverage for tests executed on JSTestDriver

1. Prepare tests manually or have tests generated by IntelliJ IDEA.
2. Create a JSTestDriver configuration file `*.jstd` or `*.conf` . For details, see JSTestDriver Configuration file .
3. Create a run configuration of the type JSTestDriver .
   To have specific files excluded from coverage analysis, specify the paths to them in the Coverage tab of the Run/Debug Configuration: JSTestDriver dialog box.
4. Start the JSTestDriver server and capture a browser to run the tests in.
5. On the main toolbar, select the JSTestDriver run configuration in the Run/Debug Configurations drop-down list and click the Run with Coverage button .
6. Monitor the code coverage in the Coverage tool window.

Measuring code coverage for tests executed on Karma

1. Prepare tests manually or have tests generated by IntelliJ IDEA.
2. Create a Karma configuration file `karma.conf` . For details, see Running Unit Tests on Karma .
3. Create a run configuration of the type Karma .
4. On the main toolbar, select the Karma run configuration in the Run/Debug Configurations drop-down list and click the Run with Coverage button .
5. Monitor the code coverage in the Coverage tool window. Note that when the tests are executed on `Karma` the Coverage tool window does not contain the Generate Coverage Report toolbar button because a coverage report is actually generated on the disk every time Karma tests are run. The format of a coverage report can be configured in the configuration file, for example:

```
// karma.conf.js
                    module.exports = function(config) {

                    config.set({ ...

                    // optionally, configure the reporter

                    coverageReporter: { type : 'html', dir
                    : 'coverage/'
                    }

                    ...

                    });};
```

The following `type` values are acceptable:

- `html` produces a bunch of HTML files with annotated source code
- `lcovonly` produces an `lcov.info` file
- `lcov` produces HTML + `.lcov` files. This format is applied by default.
- `cobertura` produces a `cobertura-coverage.xml` file for easy Hudson integration
- `text-summary` produces a compact text summary of coverage, typically to the console
- `text` produces a detailed text table with coverage for all files

Minifying JavaScript

The term minification or compression in the context of JavaScript means removing all unnecessary characters, such as spaces, new lines, comments without affecting the functionality of the source code.

These characters facilitate working with the code at the development and debugging stage by improving the code readability. However at the production stage these characters become extraneous: being insignificant for code execution, they increase the size of code to be transferred. Therefore it is considered good practice to remove them before deployment.

IntelliJ IDEA supports integration with the following JavaScript minification tools:

- YUI Compressor
- UglifyJS
- Closure Compiler

In IntelliJ IDEA, minifier configurations are called File Watchers. For each supported minifier, IntelliJ IDEA provides a predefined File Watcher template. Predefined File Watcher templates are available at the IntelliJ IDEA level. To run a minifier against your project files, you need to create a project-specific File Watcher based on the relevant template, at least, specify the path to the minifier to use on your machine.

On this page:

- Installing and Configuring the Closure Compiler Minification Tool
- Installing and Configuring the YUI Compressor JS or UglifyJS Minification Tool
- Creating a File Watcher
- Minifying the Code

Installing and Configuring the Closure Compiler Minification Tool

1. Download and install the Node.js runtime environment because the JavaScript minifier is started through Node.js.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Install and enable the File Watchers repository plugin.
   The plugin is not bundled with IntelliJ IDEA, but it is available from the IntelliJ IDEA plugin repository plugin repository . See Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins for details.
4. Download the compiled tool at http://dl.google.com/closure-compiler/compiler-latest.zip .
5. Extract the archive to the folder under the Node.js. This ensures that Node.js, which is required for starting the tool, will be specified in the path to it.

Installing and Configuring the YUI Compressor JS or UglifyJS Minification Tool

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The JavaScript minifier is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the JavaScript minifier.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the JavaScript minifier and npm from any folder.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Install and enable the File Watchers repository plugin.
   The plugin is not bundled with IntelliJ IDEA, but it is available from the IntelliJ IDEA plugin repository plugin repository . See Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins for details.
4. Download and install the JavaScript minification tool. The easiest way is to use the Node Package Manager (npm), which is a part of Node.js .
   1. Switch to the directory where the Node Package Manager (npm) is stored or define a `path` variable for it so it is available from any folder.
   2. Depending on the tool you want to use, type one of the following commands at the command line prompt:
      - To have the YUI Compressor JS installed, type:

```
npm install yuicompressor
```

      - To have the UglifyJS installed, type:

```
npm install uglify-js
```

   If you use the Node Package Manager (npm), the tools are installed under Node.js so Node.js, which is required for starting the tool, will be specified in the path to it.

Creating a File Watcher

IntelliJ IDEA provides a common procedure and user interface for creating File Watchers of all types. The only difference is in the predefined templates you choose in each case.

1. To start creating a File Watcher, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X on the main menu, and then click File Watchers under the Tools node. The File Watchers page that opens, shows the list of File Watchers that are already configured in the project.
2. Click the Add button + or press `Alt+Insert` . Depending on the tool you are going to use, choose the appropriate predefined template from the pop-up list:
   - Closure Compiler
   - UglifyJS
   - YUI Compressor JS

3. In the Program text box, specify the path to the compiler executable file or archive depending on the chosen predefined template:
   - `compiler.jar` for Closure Compiler
   - `yuicompressor-<version>.jar` for YUI Compressor JS
   - `uglifyjs.cmd` for UglifyJS

   If you installed the tool through the Node Package Manager, IntelliJ IDEA locates the required file itself and fills in the field automatically. Otherwise, type the path manually or click the Browse button ... and choose the file location in the dialog box that opens.
4. Proceed as described on page Using File Watchers .

Minifying the Code

When a Minification File Watcher is enabled (see Enabling and disabling File Watchers), minification starts automatically as soon as a file to which transpilation is applicable is changed or saved, see Configuring the behaviour of the File Watcher .

IntelliJ IDEA creates a separate file with the generated output. The file has the name of the source JavaScript file and the extension `min.js` . The location of the generated file is defined in the Output paths to refresh text box of the New Watcher dialog . However, in the Project Tree, it is shown under the source JavaScript file which is now displayed as a node.

Tracing with Spy-js

Spy-js is a tool for JavaScript developers that allows to simply debug/trace/profile JavaScript running on different platforms/browsers/devices. The tool fills gaps that existing browser development tools have and tackles common development tasks from a different angle.

The tool also traces NodeJS, server-side, applications.

On this page:

- Preparing for Tracing with Spy-js
- Spy-js Basics
- Spy-js UI
- Spy-js Tracing Session
  - Initiating a Spy-js Tracing Session
    - Launching a tracing session of a Web application
    - Launching a tracing session of a NodeJS application
  - Saving and Loading Tracing Sessions
    - To save an image of a tracing session
    - To load an image of a previous tracing session
  - Configuring the Range of Events to Display
    - Defining a new event filter
    - Choosing a filter to apply
    - Creating exclusion rules on the fly
    - Setting Timestamp Labels
  - Synchronization and Navigation between the Panes and the Editor
    - Navigating from an event or a script to the trace file
    - Navigating from an event or script to the source file
    - Navigating from a function to its call
    - Navigating through ECMAScript 6, TypeScript, or CoffeeScript
    - Advanced Trace Navigation
  - Expanding the Basic Completion List with Runtime Data (Spy-js Autocompletion)
    - Activating Spy-js autocompletion
  - Evaluating Expressions without Running a Debugging Session (Spy-js Magnification)
    - Activating Spy-js magnification
- Viewing Dependency Diagrams

Preparing for Tracing with Spy-js

1. Download and install NodeJS because it is used by the Spy-js trace server.
2. Make sure the Spy-js plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Spy-js Basics

With Spy-js, you can trace both Web applications and NodeJS applications. Although the Spy-js UI in either case is the same, the mechanisms of the tracing differ.

In order to trace a script, Spy-js has to modify it on the fly. The modification does not affect the logic of the script, it is just inserting instrumentation instructions that report back to Spy-js UI about what functions have been invoked when the script executes.

- To modify website scripts, Spy-js has to act as a proxy server that "sits" between your browser and the website you are tracing. When you open a traced website in your browser, Spy-js receives the script request, requests the script from your website, receives the script, makes the required modifications, and sends it back to your browser where the script executes, and sends the runtime information to the Spy-js UI.
  The proxy server can be configured automatically by selecting the Automatically configure system proxy check box in the Run/Debug Configuration: Spy-Js dialog or manually. See how to configure proxy settings manually on Windows , Mac , Ubuntu , iOS , Android , Windows Phone . Please note that some desktop browsers have their own screens for proxy settings configuration.
- In case of a NodeJS application, Spy-js cannot get between the NodeJS server and the scripts if the application is already running. Therefore to trace a NodeJS application, spy-js launches the NodeJS server and the application itself. This enables Spy-js to intercept and modify script requests and scripts, whereupon the tracing procedure runs as when tracing a website script.

You can also trace applications with ECMASript6, CoffeeScript, and TypeScript code: Spy-js recognizes source maps that set correspondence between the original code and the JavaScript code generated during transpilation, see Using File Watchers for details.

A Spy-js session is initiated from IntelliJ IDEA through a run configuration, see Initiating a Spy-js Session .

Spy-js UI

All the tracing-related activities, such as viewing captured events, examining their call stacks, navigating to the source code, etc. are performed in the dedicated Spy-Js Tool Window , in particular in its Trace Run Tab . The tab consists of a toolbar and three panes:

- Events Pane
  The pane shows a tree of captured events. The top-level nodes represent documents that are Web pages involved in tracing. When you hover the mouse over a document, IntelliJ IDEA displays a tooltip with the URL address of the document, the browser in which it is opened, and the operating system the browser is running on. The document node is also supplied with an icon that indicates the browser in which it is opened.

  Under each document node, events detected on the page and scripts started from it are listed. Events of the same type are grouped into visual containers. The header of a container displays the name of the events grouped in it, the average execution time across all the events within the container, and the number of events inside the container. You can expand each node and inspect the individual events in it.

Script file names have different colour indicators to help distinguishing between them when working with the Event Stack pane. By hovering your mouse over a script file name, you can see the full script URL.

Once an event is clicked, its call stack is displayed in the Event Stack pane. The stack is represented by a tree of function calls.

- Event Stack Pane

  Once an event in the Events pane is clicked, its call stack is displayed in the Event Stack pane. The stack is represented by a tree of function calls. Each tree node represents the invoked function. Node text contains the total execution time, the script file name and the function name. When you click a node, the Quick Evaluation pane shows additional function call details, parameter values and return value, occurred exception details if there was one during the function execution.

  The pane is synchronized with the editor, so you can navigate from an item in the stack tree to the corresponding trace file or source file.

  - A trace file is a write-protected prettified version of the script selected in the Events pane or the script whose function is double clicked in the Event Stack pane. A trace file is named `<file name>.js.trace`. When you double click an item in the stack tree or select it and choose Jump to Trace on the context menu of the selection, the corresponding trace file opens in the editor with the cursor positioned at the clicked function. Another approach is to press the Autoscroll to Trace toggle button and select various stack nodes. In this case, the trace file opens when you click an event or script in the Events pane.
    You can not only jump to a function but also to the place in the code where it was called from. To do that, select the required item and choose Jump to Caller on the context menu.

    The contents of the file are highlighted to display the code execution path of the selected stack node.
  - When you are tracing an application with ECMASript6, CoffeeScript, and TypeScript code, Spy-js also generates mapped trace files. These are EcmaScript 6, TypeScript, or CoffeeScript trace files with the extensions `.ts.trace`, `.coffee.trace`, or `.js.trace`. The fragments of code in these files are highlighted as if they were really executed.
  - You can also navigate to the source file displayed as is, without prettifying, by selecting an item in the Event Stack pane and choosing Jump to Source on the context menu of the selection. If the traced site is mapped with a IntelliJ IDEA project, IntelliJ IDEA detects the corresponding local file according to the mapping and opens this file in the editor. If you are tracing a site that is not mapped to a IntelliJ IDEA project, IntelliJ IDEA opens the read-only page source, just as if you chose View Page Source in the browser.
    When the traced site is mapped with a IntelliJ IDEA project, IntelliJ IDEA opens the source file on any attempt to edit the opened trace file.

- Quick Evaluation Pane

  When you click a node in the Event Stack pane, the Quick Evaluation pane shows additional function call details, parameter values and return value, occurred exception details if there was one during the function execution.

Spy-js Tracing Session

Initiating a Spy-js Tracing Session

The way you initiate a tracing session depends on the type of the application you are going to trace.

Launching a tracing session of a Web application

To a trace an application with CoffeeScript or TypeScript code, you need to transpile the original code into JavaScript, see and Transpiling TypeScript to JavaScript for details.

1. Create a run configuration of the type Spy-js. To do that, choose Run | Edit Configurations on the main menu, click the Add New Configuration toolbar button +, and choose Spy-js on the context menu. In the Run/Debug Configuration: Spy-Js dialog box that opens, specify the following:
   1. The location of the Node interpreter.
   2. The trace server port. This port number must be the same as your system proxy port. If the Automatically configure system proxy check box is selected, the specified port number is automatically set for the system proxy server. Otherwise you will have to specify the value of the field in the system proxy settings manually.
      The trace server port is filled in automatically. To avoid port conflicts, it is recommended that you accept the suggested value and keep the Automatically configure system proxy check box selected.
   3. The way to configure the proxy server.
      To have the system proxy server activated automatically with the port specified in the Trace server port field, select the Automatically configure system proxy check box.

      Clear the Automatically configure system proxy check box to specify proxy settings manually. See how to configure proxy settings manually on Windows, Mac, Ubuntu, iOS, Android, Windows Phone. Please note that some desktop browsers have their own screens for proxy settings configuration.
   4. From the Use dropdown list, choose the way to specify the way to configure a tracing session.
      - To have Spy-js apply its internal predefined configuration, choose Default configuration.
      - To have your custom manually created configuration applied, choose the Configuration file option and then specify the location of your custom configuration file in the Configuration field below.
        A configuration file is a JavaScript file with the extension `.js` or `.conf.js` that contains valid JavaScript code that meets the Spy-js configuration requirements. If IntelliJ IDEA detects files with the extension `.conf.js` in the project, these files are displayed in the drop-down list.

        Type the path to the configuration file manually or click the Browse button ... and choose the location in the dialog box that opens. Once specified, a configuration file is added to the drop-down list so you can get if next time from the list instead of specifying the path.

2. To launch a tracing session, click the Run toolbar button ▶. The Spy-Js Tool Window opens with an empty Trace Run tab and a Trace Proxy Server tab informing you about the status of the proxy server.

3. Switch to the browser and refresh the page to start debugging from. Spy-js starts capturing events on this page and the Spy-js tool window shows them in the Events pane.

Launching a tracing session of a NodeJS application

1. Create a run configuration of the type Spy-js for NodeJS. To do that, choose Run | Edit Configurations on the main menu, click the Add New Configuration toolbar button +, and choose Spy-js for NodeJS on the context menu. In the Run/Debug Configuration: Spy-js for NodeJS dialog box that opens, specify the following:
   1. The location of the Node interpreter.
   2. The Node parameters, that is, the NodeJS-specific command line options to be passed to the NodeJS executable file. For example, to enable tracing

ECMAScript 6 scripts, specify `--harmony` as a Node parameter. Note that Node.js must be version 0.11.13 or higher.

3. The working directory of the application. All references in the starting NodeJS application file, for example, includes, will be resolved relative to this folder, unless such references use full paths.
   By default, the field shows the project root folder. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button and select the location in the dialog box, that opens.
4. The JavaScript file to start the application with.
   If you are going to trace CoffeeScript, specify the path to the generated JavaScript file. The file can be generated externally or through transpilation using file watchers. For more details, see Transpiling CoffeeScript to JavaScript .
5. The application parameters: the NodeJS-specific arguments to be passed to the application start file through the process.argv array.
6. The environment variables for the NodeJS executable file, if applicable. See Run/Debug Configuration: Spy-Js for Node.js for details.
7. The configuration file with the configuration settings to apply to the tracing session.
   A configuration file is a JavaScript file with the extension `.js` or `.conf.js` that contains valid JavaScript code that meets the Spy-js configuration requirements . If IntelliJ IDEA detects files with the extension `.conf.js` in the project, these files are displayed in the drop-down list.

   Type the path to the configuration file manually or click the Browse button and choose the location in the dialog box that opens. Once specified, a configuration file is added to the drop-down list so you can get if next time from the list instead of specifying the path.
8. The trace server port. This port number must be the same as your system proxy port. If the Automatically configure system proxy check box is selected, the specified port number is automatically set for the system proxy server. Otherwise you will have to specify the value of the field in the system proxy settings manually.
   The trace server port is filled in automatically. To avoid port conflicts, it is recommended that you accept the suggested value and keep the Automatically configure system proxy check box selected.

2. To launch a tracing session, click the Run toolbar button . The Spy-js tool and the NodeJS application start. The Spy-Js Tool Window opens showing the captured events in the Trace Run tab.

Saving and Loading Tracing Sessions

You can save an image of a tracing session and load this image at any time later. The `.json` files that store the calls and properties of the session are compressed into a `zip` archive. Upon request, when you choose Load trace, the `.json` files are extracted from the archive and loaded into Spy-js.

Note that a loaded image does not restore the session because no scripts are actually executed. All you can do is analyze the flow and properties of previously executed code.

To save an image of a tracing session

– Click the button on the Events toolbar, and then choose Save trace from the list. IntelliJ IDEA compresses all the affected `.json` files in a `zip` archive and opens the folder where the archive is saved.

To load an image of a previous tracing session

1. Initiate a tracing session of the same type as the session you are going to load the image of: a Spy-js or a Spy-js for NodeJS respectively, see Initiating a Spy-js Tracing Session above. We need to have this fake session running because otherwise Spy-js would be inactive and thus the load trace functionality would be unavailable.
   Of course you can load a trace while actually tracing an application, but in this case the currently running session will be lost after loading.

2. When the session starts, click the button on the Events toolbar and choose Load Trace from the list.

3. In the dialog box that opens, choose the location of the `zip` archive with the image of the desired session. Spy-js stops the running and shows the loaded trace in a new tab named Loaded <loaded session>.
   Note that a loaded image does not restore the session because no scripts are actually executed. All you can do is analyze the flow and properties of previously executed code.

Configuring the Range of Events to Display

By default, the Spy-js tool captures all events on all opened Web pages, excluding https secure web sites, unless you have specified a URL address explicitly in the run configuration. The Events pane of the Spy-js tool window shows all captured events. If for some reasons you do not want to have all events captured, you can suppress capturing some of them by applying user-defined event filters. All the available filters are listed upon clicking the Capture Events button on the toolbar, the currently applied filter is marked with a tick. By default the Capture All predefined filter is applied. To stop capturing events without stopping the application, choose Mute All. The application is still running but the Events pane shows the last captured event. This is helpful if you want to analyze a script and therefore need it to be displayed in the Events pane instead of being removed as new events are captured. You can define new custom filters or add event patterns to existing filters on the fly.

Defining a new event filter

1. Click the Capture Events button on the toolbar, and then choose Edit Capture Exclusions from the list.

2. In the Spy-Js Capture Exclusions Dialog that opens, click the Add toolbar button in the left-hand pane.

3. In the right-hand pane, specify the filter name in the Exclusion name field and configure a list of exclusion rules.
   To add a rule, click , the Add Condition to Exclusion dialog box opens. Type a pattern in the Value/pattern text box, in the Condition type drop-down list specify whether the pattern should be applied to event types or script names. Note, that glob pattern matching is used. When you click OK, IntelliJ IDEA brings you to the Spy-Js Capture Exclusions Dialog .

   To edit a rule, select it in the list, click , and update the rule in the dialog box that opens. To remove a rule, select it in the list and click .

Creating exclusion rules on the fly

While navigating through the tree of already captured events in the Events pane, you may come across some events or scripts that you definitely do not want to trace. You can create a filter as described above but in this case you will have to leave the pane. With IntelliJ IDEA, you can create an exclusion rule based on any event or script, as soon as you have detected such event or script, right from the Events pane. The rule will be either added to the currently applied filter or a

new filter will be created if the current setting is Capture All.

- To add an event to an exclusion filter on the fly, select the event to exclude and choose Mute <event name> event or Mute <script name> file.

If a user-defined filter is currently applied, the new rule is added to it silently. If Capture All is currently active, the Spy-Js Capture Exclusions Dialog opens, where you can create a new filter based on the selected event or script or choose an existing filter and add the new rule to it.

Setting Timestamp Labels

Timestamp labels help you to analyze your code execution within a specific period of time. For example, you can set two timestamp labels and view which events were captured between them. Or on the contrary, you can locate the events that were not captured within a certain period of time although you expected them to be and thus detect performance problems.

- To set a timestamp label, choose Add Label on the context menu in the Events pane. A label Labelled at <timestamp> is added under the document node.

Synchronization and Navigation between the Panes and the Editor

The Events and Event Stack panes are synchronized: when you click an event or script in the Events pane, its call stack is displayed in the Event Stack pane. To have also the corresponding trace file opened in the editor, press the Autoscroll to Trace toggle button on the toolbar.

The Event Stack pane is synchronized with the editor: when you click an item in the stack tree twice, the corresponding trace file opens in the editor with the cursor positioned at the clicked function.

To synchronize the Events pane directly with the editor, press the Autoscroll to Trace toggle button on the toolbar. In this case, as soon as you click a node in the Events pane, its call stack is displayed in the Event Stack pane and the corresponding trace file is opened in the editor. With the Autoscroll to Trace mode turned on, when you navigate through the Event Stack the corresponding files are also automatically opened in the editor with the corresponding functions highlighted.

Navigating from an event or a script to the trace file

A trace file is a write-protected prettified version of the script selected in the Events pane or the script whose function is double clicked in the Event Stack pane. A trace file is named `<file name>.js.trace`. To navigate from an event or a script to the trace file, do one of the following:

- Activate automatic navigation:
  1. In the Events pane, press the Autoscroll to Trace toggle button on the toolbar.
  2. Click the required event or script.

- In the Event Stack pane, click the required item in the call stack twice or choose Jump to Trace. on the context menu. The trace file opens with the cursor positioned at the clicked function.

Navigating from an event or script to the source file

You can also navigate to the source file displayed as is, without prettifying.

- In the Event Stack pane, select the required item in the call stack and choose Jump to Source on the context menu of the selection.

If the traced site is mapped with a IntelliJ IDEA project, IntelliJ IDEA detects the corresponding local file according to the mapping and opens this file in the editor. If you are tracing a site that is not mapped to a IntelliJ IDEA project, IntelliJ IDEA opens the read-only page source, just as if you chose View Page Source in the browser.

Navigating from a function to its call

To navigate from a function to the place in the code where it was called from:

- In the Event Stack pane, select the required item in the call stack and choose Jump to Caller on the context menu of the selection.

Navigating through ECMAScript 6, TypeScript, or CoffeeScript

Spy-js supports source maps, which means that you can now jump from the Event Stack pane right to the original source code in ECMAScript 6, TypeScript or CoffeeScript and observe what code fragments were executed.

Spy-js also generates mapped trace files. These are EcmaScript 6, TypeScript, or CoffeeScript trace files with the extensions `.ts.trace`, `.coffee.trace`, or `.js.trace`. The fragments of code in these files are highlighted as if they were really executed.

1. Configure the way source maps are treated by clicking on the toolbar of the Events stack and choosing the following options on the context menu:
   - Choose Enable Source Map Look-up to enable navigation to the ECMAScript 6, TypeScript or CoffeeScript source code using the source maps generated during transpilation.
   - Choose Enable source map generation, to generate source maps for everything to map the instrumented code. Choose this option if you are going to debug the original code in Chrome Dev Tools or FireFox FireBug development tools.
   - Choose Always open source mapped trace if available, to have Spy-js try to open the mapped trace file when you invoke navigation from an event to its caller.

2. To navigate from a function to the source code, select the function in question in the Event Stack pane and choose one of the following options on the context menu of the selection:
   - To navigate to the ECMAScript 6, TypeScript or CoffeeScript source code, choose Jump to Source on the context menu of the selection.
   - To navigate to the JavaScript trace file, choose Jump to Trace.
   - To navigate to the mapped trace file (ECMAScript 6, TypeScript or CoffeeScript), choose Jump to Mapped Trace.

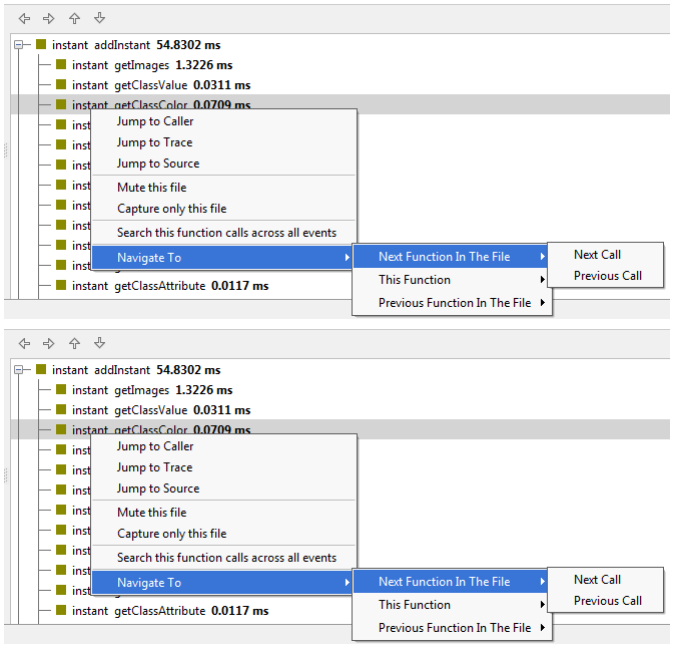
3. To navigate from a function to its call, select the function in the Event Stack and choose Jump to Caller.
   - If the Always open source mapped trace if available option is selected, the corresponding mapped trace file opens.
   - If the Always open source mapped trace if available option is not selected, the JavaScript trace file opens.

Alternatively, you can navigate to the executed JavaScript code by choosing Jump to Trace.

Advanced Trace Navigation

With advanced trace navigation, you can move through the whole stack based on calls and locate the functions that have not been called, that is, locate the fragments of code that have not been executed and analyze the reason for them to be skipped.

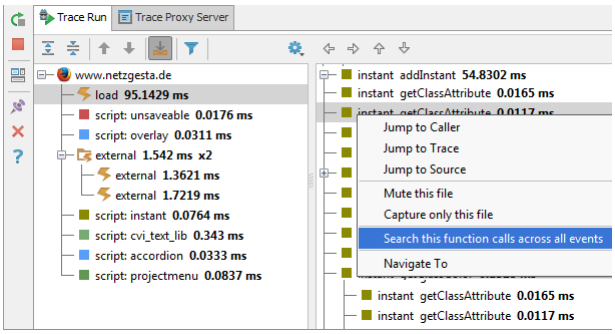
The following six actions are available: move to the next/previous call of the next/current/previous function in a trace file. The full list of actions is available from the context menu in the Event Stack pane. Moving to the next and previous calls of the selected function, to the previous call of the previous function, and to the next call of the next function are also available from the navigation toolbar of the Event Stack pane.



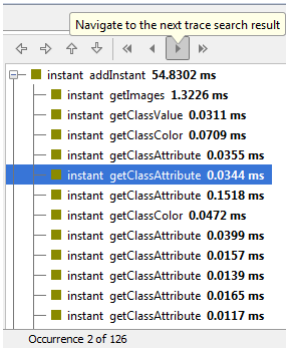
When you choose one of these actions, the cursor jumps to the call in the stack. If the Autoscroll to Trace toggle button is pressed, the corresponding trace file opens automatically with the cursor positioned at the call.

Advanced trace search lets you navigate between the calls of a function within the whole trace (across all the traced events). This means that if you are tracing 5 pages in the browser and the Events pane, accordingly, shows 5 document nodes, IntelliJ IDEA searches for the calls of the selected function under all these nodes and displays the number of found calls of the function in the Status bar.

- To invoke the search for the calls of a function in all document nodes, select the function in question in the Event Stack pane and choose Search this function calls across all events on the context menu of the selection. The number of found calls is displayed in the Status bar, and the toolbar shows four previously hidden navigation chevron buttons.



- Use the chevron buttons to navigate within the found calls:
  - To jump to the first detected call, click .
  - To jump to the last detected call, click .
  - To jump to the next detected call, click . The Status bar shows a message: `Occurrence <number> of <total number of detected calls>`
  - To jump to the previous detected call, click .



The search results are reset and the search toolbar is hidden when you invoke another advanced search or navigation.

Also keep in mind that the number of call occurrences is calculated when you choose the Search this function calls across all events option. As you analyze the

detected calls, the time passes, new events are captured, and the first detected call can happen to be already removed from the stack which means that it is no longer available for navigation.

Expanding the Basic Completion List with Runtime Data (Spy-js Autocompletion)

The term Spy-js autocompletion denotes expanding the basic completion list with suggestions retrieved from the runtime data. The Spy-js autocompletion functionality is available from source files for the code that has already been executed (highlighted green in the corresponding trace file).

When you position the caret at a symbol in the source file and press `Ctrl+Space`, Spy-js retrieves data from the browser or from the running NodeJS application and merges it with the basic completion list according to the following rules:

1. If an object both is present on the basic completion list and is retrieved from the runtime, the variant that provides more information about parameters, attributes, their type, etc. remains on the list.
2. Objects retrieved by Spy.js are shown on top of the list and marked with the icon. If a retrieved object is specific for a browser, the object is marked with the icon and with the icon of this browser.

Activating Spy-js autocompletion

By default, the functionality is turned off. To activate it:

– Click the button on the Events toolbar, and then choose Enable Spy-js autocomplete and magnifier from the list.

Evaluating Expressions without Running a Debugging Session (Spy-js Magnification)

The term Spy-js magnification denotes evaluating expressions without actually running a debugging session. When you click the expression in question or position the caret at it and press `Ctrl+Alt+F8`, a tooltip is displayed below the expression showing the expression value. If Spy-js retrieves several values, click + icon in the tooltip to expand the list of values.

The magnification functionality is available from source files for both executed and not yet executed code.

By default, the functionality is turned off.

Activating Spy-js magnification

By default, the functionality is turned off. To activate it:

– Click the button on the Events toolbar, and then choose Enable Spy-js autocomplete and magnifier from the list.

Viewing Dependency Diagrams

With Spy-js, you can build and examine runtime application/event diagrams for client-side and Node.js applications.

1. Generate a diagram:
   – To build a diagram with the dependencies within the entire application, select the document node and choose Show application dependency diagram on the context menu of the selection.
   – To build a diagram with the dependencies of a single event, select the required event in the Events pane and choose Show event dependency diagram.

2. Analyze the diagram:
   – The diagram is opened in a separate editor tab. The nodes in the diagram represent your project files, while the edges represent the fact that there's one or more functions in the source file that invoke functions in the target file.
   – To examine the details of a node or an edge, select the node or the edge in question and view its Details tree in a dedicated pane in the upper right-hand corner of the editor. The pane displays the connecting function combinations, along with event(s) the calls are made within and the number of calls made.

Using AngularJS

On this page:

- Overview
- Before You Start
- Setting Up AngularJS Support in a IntelliJ IDEA Project
  - Creating an Empty IntelliJ IDEA Project for AngularJS
  - Installing AngularJS Manually in an Empty Project
  - Installing AngularJS in an Empty Project through NPM and Bower
- Generating an AngularJS Application Stub

Overview

IntelliJ IDEA provides integration with the AngularJS framework , versions 1 and 2. This support involves:

- AngularJS-aware code completion for `ng` directives (also including custom directives), controller and application names, and code insights for data bindings inside curly-brace expressions `{{}}` .
- AngularJS-specific navigation:
  - Between the name of a controller name in HTML and its definition in JavaScript.
  - Between `ngView` or `&routeProvider` to the template.
  - Go To Symbol navigation for entities.
- Quick documentation look-up by pressing `Ctrl+Q` , see Viewing Inline Documentation for details.
- AngularJS router diagrams .

Before You Start

Make sure the NodeJS and AngularJS plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Setting Up AngularJS Support in a IntelliJ IDEA Project

You can get AngularJS support in a IntelliJ IDEA project in two ways:

- Create an empty project and install AngularJS in it. This can be done either manually, by downloading the AngularJS framework and or by means of the Bower package manager, see Installing and Removing Bower Packages ..
  If you choose manual installation, you will need to configure AngularJS as a IntelliJ IDEA JavaScript library. If you use Bower, this will be done automatically, without any steps from your side.
- Have IntelliJ IDEA generate a stub of an AngularJS-specific project. IntelliJ IDEA sets up the AngularJS-specific project structure and generates the `bower.json` and `package.json` files with the definitions of all the necessary dependencies. See Generating an AngularJS Application Stub below on this page.

Creating an Empty IntelliJ IDEA Project for AngularJS

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose Static Web and click Next.
4. On the second page of the wizard, specify the project name and the folder to create it in.
   From the Version drop-down list, choose the branch https://github.com/angular/angular.js to download the project template from. By default, `master` if shown.
5. When you click Finish , IntelliJ IDEA creates and opens an empty project.

Installing AngularJS Manually in an Empty Project

1. Download the AngularJS framework at http://angularjs.org/ .
2. Open the empty project where you will use AngularJS.
3. Configure AngularJS as a IntelliJ IDEA JavaScript library, to let IntelliJ IDEA recognize AngularJS-specific structures and provide full coding assistance:
   1. Open the Settings dialog box , and click JavaScript Libraries .
   2. In the Libraries area, click the Add button.
   3. In the New Library dialog box that opens, specify the name of the library.
   4. Click the Add button + next to the list of library files and choose Attach Files or Attach Directory on the context menu, depending of whether you need separate files or an entire folder.
   5. Select the `Angular.js` or `Angular.min.js` , or an entire directory in the dialog box that opens. IntelliJ IDEA returns to the New Library dialog box where the Name read-only field shows the name of the selected files or folder.
   6. In the Type field, specify which version you have downloaded and are going to add.
      - If you added `Angular.js` , choose Debug. This version is helpful in the development environment, especially for debugging.
      - If you added the minified `Angular.min.js` , choose Release. This version is helpful in the production environment because the file size is significantly smaller.

   Learn more at Configuring JavaScript Libraries .

Installing AngularJS in an Empty Project through NPM and Bower

1. Open the empty project where you will use AngularJS.
2. Download and install Node.js . This runtime environment is required for two reasons:
   - The Bower package manager is started through Node.js.
   - Node Package Manager (npm), which is a part of Node.js, is also the easiest way to download Bower.

   For details on using Node.js in IntelliJ IDEA, see Node.js You can also define Node.js as an external tool, as described in the section Configuring third-party tools . This approach is helpful, when you need facilities that are missing in the plugin, for example, the possibility to pass certain parameters as wildcards.

3. Install Bower as described in Using Bower Package Manager .
4. If you are going to use the command line mode, make sure the following paths are added to the PATH variable :
   1. The path to the parent folder of the Node.js executable file.
   2. The path to the `npm` folder.
   3. The path to the parent folder of the Bower executable file.

   This enables you to launch the tools from any folder.
5. Install and enable the NodeJS and AngularJS repository plugins. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.
6. Run Bower from IntelliJ IDEA using the Bower page of the Settings dialog box.
   1. Open the Settings dialog box , and click JavaScript, then click Bower.
   2. On the Bower page that opens, the Packages area shows all the Bower-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
   3. In the Available Packages dialog box that opens, select the `AngularJS` package.
   4. Specify the installation mode:
      - To have the tool installed globally at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project, select the Options check box and type `-g` in the text box next to it.
      - To have the tool installed in the current project and thus restrict its use to this project, clear the Options check box.
   5. Optionally specify the product version and click Install Package to start installation.

Generating an AngularJS Application Stub

IntelliJ IDEA can generate a project stub for developing applications using AngularJS.

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose AngularJS and click Next.
4. On the second page of the wizard, specify the project name and the folder to create it in.
   From the Version drop-down list, choose the branch https://github.com/angular/angular.js to download the project template from. By default, `master` if shown.
5. When you click Finish , IntelliJ IDEA sets up the AngularJS-specific project structure and generates the `bower.json` and `package.json` files with the definitions of all the necessary dependencies.
6. Download the AngularJs dependencies that contain the AngularJS code and the tools that support development and testing.
   1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
   2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
   3. At the command line prompt, type the following command:

      ```
      npm install
      ```

   Learn more about the installation of dependencies in the Install Dependencies section of the `readme.md` file.

Using Angular CLI

IntelliJ IDEA provides integration with Angular CLI.

On this page:

- Before You Start
- Installing Angular CLI in IntelliJ IDEA
- Generating an Angular CLI Application Stub
- Using Angular CLI Templates

Before You Start

Make sure the NodeJS and AngularJS plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Installing Angular CLI in IntelliJ IDEA

To use Angular CLI in IntelliJ IDEA projects, you need to install the `angular-cli` package globally. Do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
  3. Type the following command at the command line prompt:

     ```
     npm install -g angular-cli
     ```

     The `-g` key makes the framework run in the global mode. Because the installation is performed through NPM, the Angular CLI framework is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the framework from any folder.

     For more details on the NPM operation modes, see npm documentation . For more information about installing the Angular CLI framework, see https://npmjs.org/package/angular-cli .
- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
  3. In the Available Packages dialog box that opens, select the required package to install.
  4. Select the Options check box and type `-g` in the text box next to it.
  5. Optionally specify the product version and click Install Package to start installation.

Generating an Angular CLI Application Stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose AngularCLI and click Next.
4. On the second page of the wizard, specify the project name and the folder to create it in.
   In the Node Interpreter field, specify the NodeJS interpreter to use. Choose a configured interpreter from the drop-down list or choose Add to configure a new one, see Configuring Node.js Interpreters
5. When you click Finish , IntelliJ IDEA sets up the AngularJS-specific project structure and generates the `bower.json` and `package.json` files with the definitions of all the necessary dependencies.

Using Angular CLI Templates

The Angular CLI templates are listed on the Live Templates page of the Settings / Preferences Dialog , under the Angular2 node.

To use a template, type its abbreviation in the editor and then press Tab to expand it. Press Tab again to jump to the next edit location in the template.

Using Bower Package Manager

IntelliJ IDEA provides interface for installing, uninstalling, and upgrading client-side libraries and frameworks for your project using the Bower Package Manager . Alternatively, you can use the tool in the command line mode from the embedded local terminal .

The easiest way to install the Bower package manager is to use the Node Package Manager (npm), which is a part of Node.js . See Installing and Removing External Software Using Node Package Manager for details.

Depending on the desired location of the Bower package manager executable file, choose one of the following methods:

- Install the package manager globally at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project.
- Install the package manager in a specific project and thus restrict its use to this project.
- Install the package manager in a project as a development dependency .

In either installation mode, make sure that the parent folder of the Bower package manager is added to the `PATH` variable. This enables you to launch the package manager from any folder.

IntelliJ IDEA provides user interface both for global and project installation as well as supports installation through the command line.

On this page:

- Preparing to Install Bower
- Installing Bower Globally
- Installing Bower in a Project
- Creating a Bower Configuration File bower.json
- Configuring Bower in IntelliJ IDEA

Preparing to Install Bower

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The Bower package manager is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the Bower package manager.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Bower package manager and npm from any folder.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Installing Bower Globally

Global installation makes a package manager available at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project. Moreover, during installation the parent folder of the package manager is automatically added to the `PATH` variable, which enables you to launch the package manager from any folder. To install the package manager globally, do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
  3. Type the following command at the command line prompt:

     ```
     npm install -g bower
     ```

     The `-g` key makes the package manager run in the global mode. Because the installation is performed through NPM, the Bower package manager is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the package manager from any folder.

     For more details on the NPM operation modes, see npm documentation . For more information about installing the Bower package manager, see https://npmjs.org/package/bower .
- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click .
  3. In the Available Packages dialog box that opens, select the required package to install.
  4. Select the Options check box and type `-g` in the text box next to it.
  5. Optionally specify the product version and click Install Package to start installation.

Installing Bower in a Project

Installing a package manager in a specific project restricts its use to this project. To run project installation, do one of the following:

- Run the installation from the command line:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install bower
```

- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
  3. In the Available Packages dialog box that opens, select the required package.
  4. Optionally specify the product version and click Install Package to start installation.

Creating a Bower Configuration File bower.json

1. In the command line mode, switch to your project directory.
2. Type the following command at the command line prompt:

```
bower init
```

   If Bower does not start, check the installation: the parent folder or the Bower executable file should be specified in the PATH variable.
3. Answer the questions to specify the following basic settings:
   - The testing framework to use.
   - The browsers to be captured automatically.
   - The patterns that define the location of test files to be involved in testing or excluded from it. For more details, see

   For more details, see Bower: Configuration .

Configuring Bower in IntelliJ IDEA

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Bower under JavaScript.
2. On the Bower page that opens, specify the location of the Node.js and Bower executable files and the `bower.json` configuration file.

Installing and Removing Bower Packages

Bower packages can be installed and used only in a specific project. IntelliJ IDEA supports two installation modes: from the command line and through the dedicated user interface.

On this page:

- Installing a Bower Package in the Command-Line Mode
- Installing a Bower Package through the IntelliJ IDEA Interface
- Removing Bower Packages
- Installing a Bower Package as a Development Dependency

Installing a Bower Package in the Command-Line Mode

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
2. Switch to the directory where Bower is stored or define a `PATH` variable for it so it is available from any folder, see Installing Bower .
3. Type the following command at the command line prompt:

```
bower install <tool name>
```

Installing a Bower Package through the IntelliJ IDEA Interface

1. Run Bower from IntelliJ IDEA using the Bower page of the Settings dialog box.
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Bower under JavaScript.
   2. On the Bower page that opens, the Packages area shows all the Bower-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
   3. In the Available Packages dialog box that opens, select the required package.
   4. Optionally specify the product version and click Install Package to start installation.

Removing Bower Packages

Open the Bower page, select the package to remove, and click —.

Installing a Bower Package as a Development Dependency

If a Bower package is a documentation or a test framework, which are of no need for those who are going to re-use your application, it is helpful to have it excluded from download for the future. This is done by marking the tool as a development dependency , which actually means adding the Bower package in the `devDependencies` section of the `package.json` file.

With IntelliJ IDEA, you can have a Bower package marked as a development dependency right during installation. Do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where Bower is stored or define a `PATH` variable for it so it is available from any folder, see Installing Bower .
  3. Type the following command at the command line prompt:

```
bower install -dev <tool name>
```

- Install the package using the IntelliJ IDEA user interface:
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Bower under JavaScript.
  2. On the Bower page that opens, the Packages area shows all the Bower-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
  3. In the Available Packages dialog box that opens, select the required package.
  4. Select the Options check box and type `--dev` in the text box next to it.
  5. Optionally specify the product version and click Install Package to start installation.

After installation, a Bower package is added to the `devDependencies` section of the `package.json` file.

Using the Flow Type Checker

IntelliJ IDEA provides basic support of Flow static type checker that brings type annotations to JavaScript. This support involves recognition and syntax highlighting of Flow structures on all operating systems. If you are using Unix or OS X, you can also run Flow in the command-line mode.

For information on installing Flow, see Getting Started with Flow .

To have IntelliJ IDEA recognize Flow structures, provide correct syntax highlighting, and avoid false-positive error highlighting, change the JavaScript language level to Flow:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click JavaScript under Languages&Frameworks.
2. On the JavaScript page that opens, choose Flow from the JavaScript Language Version drop-down list.

Using Gulp Task Runner

IntelliJ IDEA provides integration with the Gulp.js Task Runner . Gulp.js support involves:

- Parsing `Gulpfile.js` files, recognizing definitions of tasks.
- Building trees of tasks.
- Navigation between a task in the tree and its definition in the `Gulpfile.js` file.
- Running and debugging tasks.
- Configuring the task execution mode and output.
- Creating a Gulp.js run configuration automatically.

On this page:

- Preparing to Install Gulp.js
- Installing Gulp.js
  - Installing Gulp.js in the command line mode
  - Installing Gulp.js through the IntelliJ IDEA interface
- Running Gulp.js Tasks
  - Running Gulp.js Tasks from the Tasks Tree
    - Building a tasks tree
    - Running a task
  - Running Tasks from Gulpfile.js
  - Running and Debugging Tasks According to a Run Configuration
    - Creating a Gulp.js run configuration
    - Running a Task according to a Run Configuration
    - Debugging Gulp Tasks
  - Running a Gulp Task as a Before-Launch Task
- Running Gulp.js Tasks Automatically

Preparing to Install Gulp.js

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The Gulp.js task runner is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the Gulp.js task runner.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Gulp.js task runner and npm from any folder.

2. Install and enable the NodeJS plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Installing Gulp.js

To run Gulp in a IntelliJ IDEA project you need a Gulp package installed locally, under the project root.

You can additionally have a global installation of Gulp. In this case, the path to the Gulp installation will be added to the `PATH` environment variable. This will allow you to invoke Gulp from any folder when working in the command line mode.

The basic Gulp.js functionality is implemented in the `gulp.js` package. The package is responsible for building project trees of Gulp tasks and for coding assistance in editing the `Gulpfile.js` file. The `gulp` package can be installed either in the command line mode or through the IntelliJ IDEA user interface. In either case, installation is performed via npm.

Installing Gulp.js in the command line mode

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).

2. Do one of the following:
   - To install Gulp.js in the current project, switch to the project root folder and type the following command at the command line prompt:

     ```
     npm install gulp
     ```

   - To install Gulp.js globally, type:

     ```
     npm install --global gulp
     ```

Installing Gulp.js through the IntelliJ IDEA interface

1. Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box: Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.

2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.

3. In the Available Packages dialog box that opens, select the `gulp` package.
   Optionally:
   - Specify the version to install.
   - To have the package installed globally, select the Options check box and type `--global` in the text box.

4. Click Install Package to start installation.

Running Gulp.js Tasks

Gulp.js tasks can be run either from the tasks tree in the dedicated Gulp Tool Window , from the `Gulpfile.js` file, by launching a Gulp.js run configuration , or as aa before-launch task from another run configuration.

Running Gulp.js Tasks from the Tasks Tree

Gulp.js starts building a tasks tree as soon as you invoke Gulp.js by choosing Show Gulp Tasks on the context menu of a `Gulpfile.js` in the Project tool window or of a `Gulpfile.js` opened in the editor. The tree is built according to the `Gulpfile.js` file on which Gulp.js was invoked. If you have several `Gulpfile.js` files in your project, you can build a separate tasks tree for each of them and run tasks without abandoning the previously built tasks trees. Each tree is shown under a separate node.

Technically, IntelliJ IDEA invokes Gulp.js and processes `Gulpfile.js` according to the default Gulp.js run configuration. However this is done silently and does not require any steps from your side.

Building a tasks tree

- If the Gulp tool window is not opened yet, do one of the following:
  - Select the required `Gulpfile.js` file in the Project tool window and choose Show Gulp Tasks on the context menu of the selection.
  - Open the required `Gulpfile.js` file in the editor and choose Show Gulp Tasks on the context menu of the editor.

  In either case, the Gulp tool window opens showing the tasks tree built according to the selected or opened `Gulp.js` file.

- If the Gulp tool window is already opened, click on the toolbar and choose the required `Gulpfile.js` file from the list. IntelliJ IDEA adds a new node and builds a tasks tree under it. The title of the node shows the path to the `Gulpfile.js` file according to which the tree is built.

- To re-build a tree, switch to the required node and click on the toolbar.

If your `Gulpfile.js` is written in ECMA6 , by default IntelliJ IDEA does not recognize this format and fails to build a tasks tree. To solve this problem, update the default Gulp.js run configuration:

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. Under the Defaults node, click Gulp.js.
3. In the Run/Debug Configuration: Gulp.Js dialog box that opens, type `--harmony` in the Node options text box and click OK.

Running a task

- To run a Gulp.js task from a tree of tasks, do one of the following:
  - To run the task, select the root node in the tree and choose on the context menu of the selection.
  - To navigate to the definition of a task, select the required task in the tree and choose on the context menu of the selection.

  The task execution output will be displayed in the Run tool window . The tool window shows the Gulp.js output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

- To run several tasks, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required tasks, then choose Run on the context menu of the selection.

Running Tasks from Gulpfile.js

You can run tasks right from the `Gulpfile.js` file opened in the editor without previously building a tree of tasks.

- Position the cursor at the definition of the task to run and choose Run <task name> on the context menu of the selection.
  IntelliJ IDEA creates and launches a temporary run configuration with the name of the selected task.

  The task execution output will be displayed in the Run tool window . The tool window shows the Gulp.js output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

- To save an automatically created temporary run configuration, position the cursor at the definition of the task for which it was created and choose Save <task name> on the context menu of the selection. Learn more in Creating and Saving Temporary Run/Debug Configurations .

Running and Debugging Tasks According to a Run Configuration

Besides using temporary run configurations that IntelliJ IDEA creates automatically, you can create and launch your own Gulp.js run configurations. For details about run configurations, see Run/Debug Configuration and Creating and Editing Run/Debug Configurations .

Creating a Gulp.js run configuration

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. Click the Add New Configuration button on the toolbar and select Gulp.js from the pop-up list.
3. In the Run/Debug Configuration: Gulp.Js dialog box that opens, specify the name of the run configuration, the tasks to run (use blank spaces as separators), the location of the `Gulpfile.js` file to retrieve the definitions of the tasks from, and the path to the `Gulp` package installed locally, under the project root.

Specify the location of the Node executable file and the NodeJS-specific options to be passed to this executable file, see Node parameters for details.

If applicable, specify the environment variables for the NodeJS executable file.

If applicable, in the Arguments field, specify the arguments for tasks to be executed with. Use the following format:

```
--<parameter_name> <parameter_value>
```

For example: `--env development`.
For details about passing task arguments, see https://github.com/gulpjs/gulp/blob/master/docs/recipes/pass-arguments-from-cli.md

Running a Task according to a Run Configuration

- Select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button. The output is displayed in the Run tool window.

Debugging Gulp Tasks

1. Create a Gulp.js run/debug configuration, see above.
2. Open the `Gulpfile.js` file in the editor and set the breakpoints in it where necessary.
3. To start a debugging session, select the required debug configuration from the list on the main tool bar and then choose Run | Debug <configuration name> on the main menu or click the Debug toolbar button.
4. In the Debug tool window that opens, analyze the suspended task execution, step through the task, etc. as described in Examining Suspended Program and Stepping Through the Program.

Running a Gulp Task as a Before-Launch Task

1. Open the Run/Debug Configurations dialog by choosing Run | Edit Configurations on the main menu, and select the required configuration from the list or create it anew by clicking + and choosing the relevant run configuration type.
2. In the dialog box that opens, click + in the Before launch area and choose Run Gulp task from the drop-down list.
3. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
   Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.

Running Gulp.js Tasks Automatically

If you have some tasks that you run on a regular basis, you can add the corresponding run configurations to a list of startup tasks. The tasks will be executed automatically on the project start-up.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Startup Tasks under Tools.
2. On the Startup Tasks page that opens, click + on the toolbar.
3. From the drop-down list, choose the required Gulp.js run configuration. The configuration is added to the list.
   If no applicable configuration is available in the project, click + and choose Edit Configurations. Then define a configuration with the required settings in the Run/Debug Configuration: Gulp.Js page that opens. When you save the new configuration it is automatically added to the list of startup tasks.

Using Grunt Task Runner

IntelliJ IDEA provides integration with the Grunt JavaScript Task Runner . Grunt support involves:

- Parsing `Gruntfile.js` files, recognizing definitions of tasks and targets.
- Building trees of tasks and targets.
- Navigation between a task or a target in the tree and its definition in the `Gruntfile.js` file.
- Running and debugging tasks and targets.
- Configuring the task execution mode and output.

On this page:

- Preparing to Install Grunt
- Installing Grunt
  - Installing the Grunt Command Line Interface
  - Installing Grunt in a project
- Running Grunt Tasks
  - Running Grunt Tasks from the Tasks Tree
    - Building a Tasks Tree
    - Running Tasks
  - Running and Debugging Tasks According to a Run Configuration
    - Creating a Grunt.js Run/Debug Configuration
    - Running a Task according to a Run Configuration
    - Debugging Grunt Tasks
  - Running a Grunt Task as a Before-Launch Task
- Running Grunt Tasks Automatically

Preparing to Install Grunt

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The Grunt task runner is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the Grunt task runner.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Grunt task runner and npm from any folder.

2. Install and enable the NodeJS plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Installing Grunt

To use Grunt in a IntelliJ IDEA project, you need two packages:

- A globally installed `grunt-cli` package (Grunt command line interface) for executing Grunt commands.
- A `grunt` package installed in the project to build the project tasks tree and provide coding assistance while editing the `Gruntfile.js` or `Gruntfile.coffee` file. Learn more about `Gruntfile.json` at http://gruntjs.com/getting-started#preparing-a-new-grunt-project .

These packages can be installed either in the command line mode or through the IntelliJ IDEA user interface. In either case, installation is performed via npm.

Installing the Grunt Command Line Interface

The Grunt Command Line Interface is implemented in the `grunt-cli` package which should be installed globally to make the functionality available in any IntelliJ IDEA project and from any folder. Do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
  3. Type the following command at the command line prompt:

     ```
     npm install -g grunt-cli
     ```

     The `-g` key makes the task runner run in the global mode. Because the installation is performed through NPM, the Grunt task runner is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the task runner from any folder.

     For more details on the NPM operation modes, see npm documentation . For more information about installing the Grunt task runner, see https://npmjs.org/package/grunt-cli .

- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click .
  3. In the Available Packages dialog box that opens, select the required package to install.
  4. Select the Options check box and type `-g` in the text box next to it.
  5. Optionally specify the product version and click Install Package to start installation.

Installing Grunt in a project

At the project level, the basic Grunt functionality is implemented in the `grunt` package. The package is responsible for building project trees of Grunt tasks and for coding assistance in editing the `Gruntfile.js` file. Do one of the following:

- Run the installation from the command line:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install grunt
```

- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
  3. In the Available Packages dialog box that opens, select the required package.
  4. Optionally specify the product version and click Install Package to start installation.

Running Grunt Tasks

Grunt tasks are launched in three ways:

- From tasks trees in the dedicated Grunt Tool Window which opens when you invoke Grunt from a `Gruntfile.js` file. As soon as you invoke Grunt, it starts building a tree of tasks according to the `Gruntfile.js` on which it was invoked. If a task has targets , the task is displayed as a node and the targets are listed under it.
- According to a dedicated run configuration, see Run/Debug Configuration: Grunt.Js .
- As a before-launch task, from another run configuration.

The result of executing a task is displayed in the Run tool window . The tool window shows the Grunt output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

Running Grunt Tasks from the Tasks Tree

Running Grunt tasks right from the tasks tree is easy and fast, the only disadvantage of this approach is that such important additional options as force execution and verbose mode are not available in this mode. As a result, you cannot have IntelliJ IDEA, for example, ignore warnings or provide a detailed log.

Building a Tasks Tree

- If the Grunt tool window is not opened yet, do one of the following:
  - Select the required `Gruntfile.js` file in the Project tool window and choose Show Grunt Tasks on the context menu of the selection.
  - Open the required `Gruntfile.js` file in the editor and choose Show Grunt Tasks on the context menu of the editor.

  In either case, the Grunt tool window opens showing the tasks tree built according to the selected or opened `Gruntfile.js` file.

- If the Grunt tool window is already opened, click on the toolbar and choose the required `Gruntfile.js` file from the list. IntelliJ IDEA adds a new node and builds a tasks tree under it. The title of the node shows the path to the `Gruntfile.js` file according to which the tree is built.

- To re-build a tree, switch to the required node and click on the toolbar.

Running Tasks

- To run a task or a target, do one of the following:
  - To run the task, select the root node in the tree and choose on the context menu of the selection.
  - To navigate to the definition of a task or target, select the required task or target in the tree and choose on the context menu of the selection.

  The task or target execution output will be displayed in the Run tool window . The name of the target is shown in the format `<task name>:<target name>` . The tool window shows the Grunt output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

- To run several tasks or targets, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required tasks or targets, then choose Run on the context menu of the selection.

Running and Debugging Tasks According to a Run Configuration

Besides using temporary run configurations that IntelliJ IDEA creates automatically, you can create and launch your own Grunt.js run configurations. For details about run configurations, see Run/Debug Configuration and Creating and Editing Run/Debug Configurations .

Creating a Grunt.js Run/Debug Configuration

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.

2. Click the Add New Configuration button + on the toolbar and select Grunt.js from the pop-up list.

3. In the Run/Debug Configuration: Grunt.Js dialog box that opens, specify the name of the run configuration, the tasks to run (use blank spaces as separators), the location of the `Gruntfile.js` file to retrieve the definitions of the tasks from, and the path to the `Grunt` package installed locally, under the project root. Specify the location of the Node executable file and the NodeJS-specific options to be passed to this executable file, see Node parameters for details.

If applicable, specify the environment variables for the NodeJS executable file.

To have Grunt ignore warnings and continue executing the launched task until the task is completed successfully or an error occurs, select the --f check box. Otherwise, the task execution is stopped by the first reported warning. To have the verbose mode applied and thus have a full detailed log of a task execution displayed, select the --v check box.

Running a Task according to a Run Configuration

Select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button . The output is displayed in the Run tool window .

Debugging Grunt Tasks

1. Create a Grunt.js run/debug configuration, see above .
2. Open the `Gruntfile.js` file in the editor and set the breakpoints in it where necessary.
3. To start a debugging session, select the required debug configuration from the list on the main tool bar and then choose Run | Debug <configuration name> on the main menu or click the Debug toolbar button .
4. In the Debug tool window that opens, analyze the suspended task execution, step through the task, etc. as described in Examining Suspended Program and Stepping Through the Program .

Running a Grunt Task as a Before-Launch Task

1. Open the Run/Debug Configurations dialog by choosing Run | Edit Configurations on the main menu, and select the required configuration from the list or create it anew by clicking + and choosing the relevant run configuration type.
2. In the dialog box that opens, click + in the Before launch area and choose Run Grunt task from the drop-down list.
3. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
   Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.

Running Grunt Tasks Automatically

If you have some tasks or targets that you run on a regular basis, you can add the corresponding run configurations to a list of startup tasks. The tasks will be executed automatically on the project start-up.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Startup Tasks under Tools.
2. On the Startup Tasks page that opens, click + on the toolbar.
3. From the drop-down list, choose the required Grunt run configuration. The configuration is added to the list.
   If no applicable configuration is available in the project, click + and choose Edit Configurations. Then define a configuration with the required settings in the Run/Debug Configuration: Grunt.Js page that opens. When you save the new configuration it is automatically added to the list of startup tasks.

Using JavaScript Code Quality Tools

IntelliJ IDEA provides facilities to run JavaScript-specific code quality inspections through integration with the following code verification tools:

- JSLint. The tool is bundled with IntelliJ IDEA.
- JSHint. IntelliJ IDEA comes bundled with the default version of the tool, but you can have an alternative version downloaded.
- Closure Linter. The tool is not bundled with IntelliJ IDEA, so you need to download it yourself.
- JSCS. The tool is not bundled with IntelliJ IDEA, but you can install it from the IntelliJ IDEA interface using the Node Package Manager (npm).
- ESLint. The tool is not bundled with IntelliJ IDEA, but you can install it from the IntelliJ IDEA interface using the Node Package Manager (npm).

All these tools register themselves as IntelliJ IDEA code inspections: they check JavaScript code for most common mistakes and discrepancies without running the application. When a tool is activated, it launches automatically on the edited JavaScript file. Discrepancies are highlighted and reported in pop-up information windows, a pop-up window appears when you hover the mouse pointer over a stripe in the Validation sidebar. You can also press `Alt+Enter` to examine errors and apply suggested quick fixes. Learn more about inspections and intention actions at Code Inspection and Intention Actions.

On this page:

- Enabling and Configuring the Built-in JSLint Tool
- Enabling and Configuring JSHint
- Enabling and Configuring Closure Linter
- Using JSCS
- Using ESLint

Enabling and Configuring the Built-in JSLint Tool

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.
   Expand the JavaScript node under Languages&Frameworks, and then click JSLint under Code Quality Tools.
2. On the JSLint page that opens, select the Enable check box. After that all the controls in the page become available.
3. Define the set of common mistakes to check the code for. To enable a validation, select the check box next to it.

Enabling and Configuring JSHint

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.
   Expand the JavaScript node under Languages&Frameworks, and then click JSHint under Code Quality Tools.
2. On the JSHint page that opens, select the Enable check box. After that all the controls in the page become available.
3. From the Version drop-down list, choose the version of the tool to apply.
   IntelliJ IDEA comes bundled with version 1.0.0, which is used by default. IntelliJ IDEA provides the ability to download another version, which is not bundled. Actually, the alternative version is downloaded only once, whereupon it is available without download.
4. Configure the behaviour of JSHint:
   - To have the code verified according to the rules from a previously created configuration file, select the Use config files check box.
     A configuration file is a JSON file with the extension `.jshintrc` that specifies which JSHint options should be enabled or disabled. IntelliJ IDEA will look for a `.jshintrc` file in the working directory. If the search fails, IntelliJ IDEA will search in the parent folder, then again in the parent folder. The process is repeated until IntelliJ IDEA finds a `.jshintrc` or reaches the project root. To have IntelliJ IDEA still run verification in this case, specify the default configuration file to use.

     Starting with IntelliJ IDEA version 6.0.1, the globals setting in `.jshintrc` is supported. For earlier versions, a Predefined (, separated)" tree node is displayed in the bottom of the tree in the UI.

   - To configure verification manually, clear the check box and define the set of common mistakes to check the code for in the Options area. The controls in the area fall into two groups:
     - Enforcing options: select the check boxes in this group to enable very strict behaviour of the verification tool and thus allow only safe JavaScript.
     - Relaxing options: select/clear the check boxes in this area to suppress warnings when certain types of discrepancies are detected.
     - Environments: select/clear these check boxes to specify for which environments you want global properties predefined.

     To enable a validation, select the check box next to it.

Enabling and Configuring Closure Linter

1. Download and install the Closure Linter tool.
2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.
   Expand the JavaScript node under Languages&Frameworks, and then click Closure Linter under Code Quality Tools.
3. On the Closure Linter page that opens, select the Enable check box. After that all the controls in the page become available.
4. Specify the path to the Closure Linter executable file:
   - `<Python_home>\Scripts\jslint.exe` for Windows
   - `/usr/local/bin/gjslint` for Linux and OS X
5. Specify the path to the previously created configuration file.

Note that the behaviour of Closure Linter may slightly differ depending on whether it is run from IntelliJ IDEA or in the command line mode if a configuration file is parsed before the `gjslint` process started and all the parsed options are passed as arguments to the `gjslint` process. When Closure Linter is running inside IntelliJ IDEA, the `gjslint` process is launched in the following way:

```
/path/to/gjslint --flagfile /path/to/config_file --recurse=no /path/to/user_source_file.js
```

Therefore, the following command fails if the configuration file has a space instead of a `=`:

```
--jslint_error indentation
```

Using JSCS

The JSCS tool is run through NodeJS, therefore make sure the NodeJS framework is downloaded and installed on your computer. The framework also contains the Node Package Manager(npm) through which JSCS is installed.

Integration with NodeJS and NPM is supported through the NodeJs plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

To install JSCS:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the `jscs` package and click Install Package.
   Learn more about installing tools through NPM in Installing and Removing External Software Using Node Package Manager .

To activate and configure JSCS:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.
   Expand the JavaScript node under Languages&Frameworks, and then click JSCS under Code Quality Tools.
2. Select the Enable check box to activate JSCS. After that the controls in the dialog box become available.
3. In the Node Interpreter field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens.
4. In the JSCS Package field, specify the location of the `jscs` package installed in the current project, see Installing JSCS .
5. Appoint the configuration to use.
   By default, IntelliJ IDEA first looks for a `jscsConfig` property in the `package.json` file of the current project. If no such property is found, IntelliJ IDEA looks for a `.jscsrc` or a `.jscs.json` configuration file. IntelliJ IDEA starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. Accordingly, you have to define the configuration to apply either as a `jscsConfig` property in the `package.json` file or in a `.jscsrc` or a `.jscs.json` configuration file, or in a custom JSON configuration file.

   You can also apply a predefined set of rules, either independently or in combination with a configuration file. In the latter case, the rules from the configuration file override the predefined rules.
   - To have IntelliJ IDEA look for a `jscsConfig` property in the `package.json` file or for a `.jscsrc` or a `.jscs.json` file, choose the Search for config(s) option.
   - To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the ... button and select the relevant file from the dialog box that opens.
   - To have a predefined set or rules applied, choose the desired set from the Code Style Preset drop-down list.
6. If necessary, from the Code Style Preset drop-down list, choose the set of predefined rules associated with the code style you use.

Using ESLint

The ESLint tool is run through NodeJS, therefore make sure the NodeJS framework is downloaded and installed on your computer. The framework also contains the Node Package Manager(npm) through which JSCS is installed.

Integration with NodeJS and NPM is supported through the NodeJS plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

To install ESLint:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the `eslint` package and click Install Package.
   Learn more about installing tools through NPM in Installing and Removing External Software Using Node Package Manager .

To activate and configure ESLint:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.
   Expand the JavaScript node under Languages&Frameworks, and then click ESLint under Code Quality Tools.
2. Select the Enable check box to activate ESLint. After that the controls in the dialog box become available.
3. In the Node Interpreter field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens.
4. In the ESLint Package field, specify the location of the `eslint` package installed in the current project, see Installing ESLint .
5. Appoint the configuration to use.
   By default, IntelliJ IDEA first looks for a `.eslintrc` configuration file. IntelliJ IDEA starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. If no `.eslintrc` file is found, ESLint uses its default embedded configuration file. Accordingly, you have to define the configuration to apply either in a `.eslintrc` configuration file, or in a custom JSON configuration file, or rely on the default embedded configuration.
   - To have IntelliJ IDEA look for a `.eslintrc` file, choose the Search for .eslintrc option. If no `.eslintrc` file is found, the default embedded configuration file will be used.
   - To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the ... button and select the relevant file from the dialog box that opens.
6. If necessary, in the Additional Rules Directory field, specify the location of the files with additional code verification rules. These rules will be applied after the rules from `.eslintrc` or the above specified custom configuration file and accordingly will override them.
7. If necessary, in the Extra ESLint Options field, specify additional command line options to run ESLint with using spaces as separators. See ESLint Command Line Interface Options for details.

Using Meteor

IntelliJ IDEA provides integration with the Meteor framework . Meteor support in IntelliJ IDEA involves:

- Automatic recognition of Meteor projects by detecting the `.meteor` folder and excluding the `.meteor/local` folder from project.
- Attaching the predefine Meteor library to the project automatically.
- Coding assistance: syntax highlighting, code completion, resolving references.
- Support of Spacebars via Handlebars with completion for if and each directives. There is also support for navigation between JavaScript source code and templates, allowing for easy navigation by using `Ctrl+B` to go to Declaration .
- Running and debugging the server and the client sides of applications.

On this page:

- Before You Start
- Installing Meteor
- Preparing to Develop a Meteor Application
  - Generating a Meteor Application Stub
  - Enabling Meteor Integration in an Existing Project
  - Importing Meteor Packages
- Running and Debugging a Meteor Application
  - Creating a Meteor Run Configuration
  - Configuring the Update Policy in the Debugging Mode
    - Configuring the Update Policy for the Server Side Code
    - Configuring the Update Policy for the Client Side Code
  - Running a Meteor Application
  - Debugging a Meteor Application

Before You Start

Make sure the Meteor and the Handlebars/Mustache plugins are activated. The Handlebars/Mustache enables IntelliJ IDEA to recognize Spacebars templates that are an extension of the Handlebars/Mustache templates. As a side effect, HTML files in Meteor projects are marked with the Handlebars/Mustache icon. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Installing Meteor

The installation procedure depends on the operating system you are using:

1. To install Meteor on Windows, download the `LaunchMeteor.exe` installer at https://github.com/meteor/meteor/wiki/Preview-of-Meteor-on-Windows .
2. To install Meteor on OSX (OSX 10.6 or higher is required) or Linux, type the following command:

```
$ curl https://install.meteor.com | /bin/sh
```

Learn more at http://docs.meteor.com/ .

Preparing to Develop a Meteor Application

To start your development, you need a IntelliJ IDEA project with the Meteor-specific structure. You can have an application stub that meets these requirements generated automatically or open an existing Meteor project in IntelliJ IDEA and configure Meteor support in it.

Generating a Meteor Application Stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose Meteor JS App and click Next.
4. On the second page of the wizard:
   1. Specify the project name and the folder to create it in.
   2. Specify the location of the Meteor executable file (see Installing Meteor).
   3. From the Template drop-down list, choose the sample to generate. To have a basic project structure generated, choose the Default option.

When you click Finish, IntelliJ IDEA generates a skeleton of a Meteor application, including an HTML file, a JavaScript file, a CSS file, and a `.meteor` folder with subfolders. The `.meteor/local` folder, which is intended for storing the built application, is automatically marked as excluded and is not involved in indexing.

By default, excluded files are shown in the project tree. To have the `.meteor/local` folder hidden, click the button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.

IntelliJ IDEA also automatically attaches the predefined Meteor library to the project, thus enabling syntax highlighting, resolving references, and code completion. See Configuring JavaScript Libraries for details.

Meteor uses Spacebars templates that are an extension of the Handlebars/Mustache templates. IntelliJ IDEA recognizes Spacebars templates, but as a side effect marks HTML files in Meteor projects with the Handlebars/Mustache icon.

Enabling Meteor Integration in an Existing Project

IntelliJ IDEA automatically detects Meteor projects by locating the `.meteor` folder. The `.meteor/local` folder, which is intended for storing the built application, is automatically marked as excluded and is not involved in indexing.

1. Open the desired Meteor project in IntelliJ IDEA by choosing File | Open on the main menu or clicking Open on the Welcome Screen.
2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Meteor under JavaScript.
3. On the Meteor page that opens:

1. Check the location of the Meteor executable file or specify the path to it if IntelliJ IDEA has not detected the executable file automatically.
2. To have IntelliJ IDEA involve the `.meteor/local` folder and its contents in indexing, clear the Automatically exclude ".meteor/local" directory on open project check box. By default, the check box is selected and the `.meteor/local` folder, which is intended for storing the built application, is automatically marked as excluded and is not involved in indexing.
   By default, excluded files are shown in the project tree. To have the `.meteor/local` folder hidden, click the button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.
3. Make sure the Automatically import Meteor packages as external library check box is selected.
   - When the check box is selected, IntelliJ IDEA automatically imports the external packages from the `meteor/packages` file. As a result, IntelliJ IDEA provides full range coding assistance: resolves references to Meteor built-in functions, for example, `check(true)`, and to functions from third-party packages, provides proper syntax and error highlighting, supports debugging with source maps, etc.
   - When this check box is cleared, IntelliJ IDEA does not automatically import the external packages from the `meteor/packages` file. As a result no coding assistance is provided. To improve the situation, open the `meteor/packages` file in the editor and click the Import packages as library link or run the `meteor --update` command.

   By default, the check box is selected.
4. Click Apply to save the settings without leaving the dialog box.

4. Check that IntelliJ IDEA has attached the Meteor library to the project:
   1. Click Libraries under JavaScript.
   2. On the JavaScript. Libraries page that opens, make sure the check box next to the Meteor project library in the Libraries list is selected.
   3. Click OK.

   Alternatively, choose Use JavaScript Library on the context menu of the editor and make sure a tick is set next to Meteor project library.

By default, excluded files are shown in the project tree. To have the `.meteor/local` folder hidden, click the button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.

IntelliJ IDEA also automatically attaches the predefined Meteor library to the project, thus enabling syntax highlighting, resolving references, and code completion. See Configuring JavaScript Libraries for details.

Importing Meteor Packages

Besides the predefined Meteor library that ensures basic Meteor-specific coding assistance, you can download additional packages that are defined in the `.meteor/local/packages` file.

1. Open the `.meteor/local/packages` file in the editor.
2. Click the Import Meteor Packages link in the upper right-hand corner of the screen.
3. In the dialog box that opens, specify the packages to download depending on the type of the application you are going to develop in your project.
   - Client
   - Server
   - Cordova: choose this option to import the packages that support development of Meteor applications for iOS and Android, see Meteor Cordova Phonegap Integration for details.

Running and Debugging a Meteor Application

With IntelliJ IDEA, you can debug both the client-side and the server-side of Meteor JavaScript code simultaneously using the IntelliJ IDEA debugger. The debugger also pauses at the breakpoints set in the sources stored in the `/packages` folder. This functionality is supported both for the client side and for the server side code.

Meteor applications are launched only through a dedicated run configuration. Technically, several Meteor projects that implement different applications can be combined within one single IntelliJ IDEA project. To run and debug these applications independently, create a separate run configuration for each of them with the relevant working directory. To avoid port conflicts, these run configurations should use different ports. In the Program Arguments field, specify a separate port for each run configuration in the format `--port=<port_number>`. The results are shown in the console and/or in the browser depending on the settings specified in the Browser / Live Edit tab, see Creating a run configuration below.

In the debugging mode, IntelliJ IDEA provides the Live Edit functionality which supports both automatic and manual upload of updated files on the server side. For the client side, Live Edit does not provide any way to apply the changes. For information on enabling and configuring the Live Edit functionality, see Live Editing of HTML, CSS, and JavaScript .

Creating a Meteor Run Configuration

Technically, IntelliJ IDEA creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated Meteor run configuration.

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button, and choose Meteor on the context menu.

2. In the Configuration tab of the Run/Debug Configuration: Meteor dialog box that opens, specify the name of the configuration and the location of the Meteor executable file according to the installation (see Installing Meteor).
   Optionally, in the Program Arguments field, specify the command line additional parameters to be passed to the executable file on start up, if applicable. These can be, for example, `--dev`, `--test`, or `--prod` to indicate the environment in which the application is running (development, test, or production environments) so different resources are loaded on start up.

   In the Working Directory field, specify the folder under which the application files to run are stored. This folder must contain a `.meteor` folder in the root to be treated a Meteor project. By default, the field shows the path to the IntelliJ IDEA project root.

   Technically, several Meteor projects that implement different applications can be combined within one single IntelliJ IDEA project. To run and debug these applications independently, create a separate run configuration for each of them with the relevant working directory. To avoid port conflicts, these run configurations should use different ports. In the Program Arguments field, specify a separate port for each run configuration in the format `--port=<port_number>`.

3. Switch to the Browser / Live Edit tab and configure the behaviour of the browser:

1. To view the results of the client-side code execution during running and debugging, select the After Launch check box and choose the browser to open from the drop-down list.
2. In the text box below, specify the URL address to open the application in. The default value is `http://localhost:3000`.
3. To enable debugging the client-side code in the browser, select the With JavaScript Debugger check box.
4. Click OK to save the run/debug configuration.

Configuring the Update Policy in the Debugging Mode

In the debugging mode, IntelliJ IDEA provides the Live Edit functionality which supports both automatic and manual upload of updated files on the server side. For the client side, Live Edit does not provide any way to apply the changes.

For information on enabling and configuring the Live Edit functionality, see Live Editing of HTML, CSS, and JavaScript .

The client-side code can be updated through the native Meteor hot code pushes functionality .

Configuring the Update Policy for the Server Side Code

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Debugger node under Build, Execution, Deployment, and then click Live Edit. The Live Edit page that opens.

2. In the Update area, configure the way changes made to the code during a debugging session are applied. This functionality is available only for the server-side code. To have the client side code updated, select the Enable Meteor 'Hot code push' check box on the Meteor page of the Settings dialog box.
   Note that an update will be performed only if none of the modified files have any syntax errors.
   - Auto in (ms): Choose this option to have the changes applied automatically at certain time interval and specify the delay in ms in the text box. This policy is not available for the client-side code of Meteor applications.
   - Manual: Choose this option to apply the changes manually by clicking the Update <run configuration name> JavaScript button or the Update <run configuration name> button on the toolbar of the Debug tool window.
     It is recommended that you choose this option because applying changes to the client-side code is not supported.
   - Restart if hotswap fails:
     With changes in HTML, CSS, and JavaScript on the client side, the contents of a Web page in the browser are updated without reloading. For NodeJS or Meteor applications, IntelliJ IDEA first tries to update the application incorporating the changes without restarting the NodeJS server.
     - Select this check box to have IntelliJ IDEA try restarting the server if the changes cannot be applied automatically. After the restart, IntelliJ IDEA brings you to the execution point where the problem took place.
       If even with this option chosen automatic upload still fails, tou will have to restart the server manually by clicking the Rerun <run configuration name> button .
     - When the check box is cleared, IntelliJ IDEA just displays a pop-up window informing you about the failure and suggesting you to restart the server manually.

Configuring the Update Policy for the Client Side Code

The client-side code can be updated through the native Meteor hot code pushes functionality . The functionality is enabled by default, to activate it:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Meteor under JavaScript.

2. On the Meteor page that opens, select the Enable Meteor 'Hot code push' check box.

Running a Meteor Application

- To run an application, select the required run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button .
  If in the run configuration the browser is configured to open automatically upon application start, the browser opens showing the result of application execution. You can view the same result by opening the browser of your choice manually at the URL specified in the run configuration.

Debugging a Meteor Application

1. Set the breakpoints in the code, where necessary.

2. To initiate a debugging session, select the required debug configuration from the list on the main tool bar and then choose Run | Debug <configuration name> on the main menu or click the Debug toolbar button . The Debug tool window opens showing two tabs: one for debugging the server-side code marked with and the other for debugging the client-side code marked with .

3. Explore the suspended program and step through the program .

4. Apply the changes to the application depending on the update policy defined in the run configuration:
   - To apply the changes to the client-side code, switch to the tab and click the Update <run configuration name> JavaScript button on the toolbar.
   - The changes to the server-side code are applied depending on the update policy defined in the run configuration:
     - If you have chosen manual upload, switch to the tab and click the Update <run configuration name> button on the toolbar.
     - If you have chosen automatic upload, updates to the server-side code are applied automatically. If automatic upload fails, restart the application by clicking Rerun <run configuration name> button . Choosing the Restart if hotswap fails option on the Live Edit page may help: IntelliJ IDEA will attempt to restart the server automatically.

Using PhoneGap/Cordova

In IntelliJ IDEA, you can develop applications intended for running on various mobile platforms, including Android , using the PhoneGap , Apache Cordova , and Ionic frameworks.

On this page:

- Before You Start
- Installing PhoneGap/Cordova/Ionic
  - Installing PhoneGap in the Command Line Mode
  - Installing PhoneGap/Cordova/Ionic Through the IntelliJ IDEA Interface
- Preparing to Use PhoneGap/Cordova/Ionic in a Project
  - Generating a PhoneGap/Cordova/Ionic Application Stub
  - Enabling PhoneGap/Cordova/Ionic Integration in an Existing Project
- Creating and Launching a PhoneGap/Cordova/Ionic Run Configuration

Before You Start

1. Make sure the PhoneGap/Cordova and the NodeJS plugins are enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.
2. Download and install Node.js because NPM, which is a part of the framework, is also the easiest way to download PhoneGap and Cordova.

Installing PhoneGap/Cordova/Ionic

To use PhoneGap/Cordova, you need a `phonegap` , or `cordova` , or `ionic` package. The package can be installed either in the command line mode or through the IntelliJ IDEA user interface. In either case, installation is performed via npm.

Installing PhoneGap in the Command Line Mode

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
2. Type one of the following commands at the command line prompt:

```
npm install -- global phonegap
npm install -- global cordova
npm install -- global ionic
```

With the `--global` option, the package will be installed globally so it is accessible from all your IntelliJ IDEA projects.

Installing PhoneGap/Cordova/Ionic Through the IntelliJ IDEA Interface

1. Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box: Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the `phonegap` , or `cordova` , or `ionic` package, depending on your preferences and workflow. Optionally:
   - Specify the version to install.
   - To have the package installed globally so it is accessible from all your IntelliJ IDEA projects, select the Options check box and type `--global` in the text box.
4. Click Install Package to start installation.

Preparing to Use PhoneGap/Cordova/Ionic in a Project

To start your development, you need a IntelliJ IDEA project with the PhoneGap/Cordova/Ionic-specific structure. You can have an application stub that meets these requirements generated automatically or open an existing PhoneGap/Cordova/Ionic project in IntelliJ IDEA and configure PhoneGap/Cordova/Ionic support in it.

Generating a PhoneGap/Cordova/Ionic Application Stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose Static Web in the left-hand pane.
3. In the right-hand pane, choose PhoneGap/Cordova App and click Next.
4. On the second page of the wizard, specify the project name and the folder to create it in. Specify the location of the executable file `phonegap.cmd` , or `cordova.cmd` , or `ionic.cmd` (see Installing PhoneGap/Cordova/Ionic).

When you click Finish, IntelliJ IDEA generates a skeleton of a PhoneGap/Cordova/Ionic application with the framework-specific structure.

Enabling PhoneGap/Cordova/Ionic Integration in an Existing Project

1. Open the desired PhoneGap/Cordova/Ionic project in IntelliJ IDEA by choosing File | Open on the main menu or clicking Open on the Welcome Screen.
2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click PhoneGap/Cordova under JavaScript.

3. On the PhoneGap/Cordova page that opens:
   1. Check the location of the executable file `phonegap.cmd`, or `cordova.cmd`, or `ionic.cmd` or specify the path to it if IntelliJ IDEA has not detected the executable file automatically.
      IntelliJ IDEA detects the installed version and displays it in the PhoneGap/Cordova Version read-only field,
   2. In the PhoneGap/Cordova Working Directory field, specify the folder under which the PhoneGap/Cordova/Ionic application files to run are stored.
   3. In the Plugins area, configure a list of plugins to use in your development by installing required packages. The list shows all the PhoneGap/Cordova/Ionic plugins that are currently installed on your computer, both at the global and at the project level.
      - To install a plugin, click the Install button. In the Available Packages dialog box that opens, select the required package.
        To have the plugin installed globally so it is accessible from all your IntelliJ IDEA projects, select the Options check box and type `--global` in the text box. Click Install Package.
      - To remove a plugin, select it in the list and click the Uninstall button.
      - To upgrade a plugin to the latest available version, select the plugin in the list and click the Upgrade button.

      See Apache Cordova Plugins and PhoneGap Plugins for information about plugins and their use.

Creating and Launching a PhoneGap/Cordova/Ionic Run Configuration

PhoneGap/Cordova/Ionic applications are executed according to a dedicated run/debug configuration.

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button, and choose PhoneGap/Cordova on the context menu.
2. In the Run/Debug Configuration: PhoneGap/Cordova dialog box that opens, specify the following:
   1. The name of the configuration.
   2. In the PhoneGap/Cordova Executable Path field, specify the location of the executable file `phonegap.cmd`, `cordova.cmd`, or `ionic.cml` (see Installing PhoneGap/Cordova/Ionic).
   3. In the PhoneGap/Cordova Working Directory field, specify the folder under which the PhoneGap/Cordova/Ionic application files to run are stored.
   4. From the Command drop-down list, choose the command to run. The contents of the drop-down list, depend on the actually used framework, namely, on the executable file specified in the PhoneGap/Cordova Executable Path field. The available options are:
      - For PhoneGap:
        - emulate
        - run
        - prepare
        - serve
        - remote build
        - remote run

        See https://www.npmjs.org/package/phonegap for a list of PhoneGap-specific commands with descriptions.
      - For Cordova:
        - emulate
        - run
        - prepare
        - serve

        See https://www.npmjs.org/package/cordova for a list of Cordova-specific commands with descriptions.
      - For Ionic:
        - emulate
        - run
        - prepare
        - serve

        See https://www.npmjs.org/package/ionic for a list of Ionic-specific commands with descriptions.
   5. From the Platform drop-down list, choose the platform for running on which the application is intended. The available options are:
      - Android
      - ios To emulate this platform, you need to install the ios-sim command line tool globally. You can do it through the Node Package Manager (npm), see Installing and Removing External Software Using Node Package Manager or by running the `sudo npm install ios-sim -g` command, depending on your oeprating system.
      - amazon-fireos
      - firefoxos
      - blackberry10
      - ubuntu
      - wp8
      - windows8
      - browser

      Learn more about targeted platforms at http://docs.phonegap.com/en/edge/guide_platforms_index.md.html#Platform%20Guides and http://cordova.apache.org/docs/en/4.0.0/guide_cli_index.md.html#The%20Command-Line%20Interface .
   6. For Cordova and Ionic, specify the targeted virtual or physical Android device to run the application on: select the Specify Target check box and select the required device from the drop-down list. The list shows all the virtual and physical devices that are currently configured on our machine. See http://docs.phonegap.com/en/edge/guide_platforms_android_index.md.html#Android%20Platform%20Guide for details.
3. To run a PhoneGap/Cordova/Ionic application, select the required run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button.

Using ReactJS in JavaScript and TypeScript

ReactJS introduces JSX and TSX, an XML-like syntax that you can use inside your JavaScript and TypeScript code respectively.

On this page:

- Installing ReactJS
- Enabling ReactJS Support in JavaScript Code
- Enhancing Code Completion with Typed Parameter Information
- Using ReactJS with Typescript

Installing ReactJS

Download the ReactJS Starter Kit and extract the files from the archive.

Enabling ReactJS Support in JavaScript Code

IntelliJ IDEA recognizes JSX code and provides syntax highlighting, code completion, navigation, and code analysis for it. Code completion and navigation is also provided for ReactJS methods. See Auto-Completing Code and Navigating to Declaration or Type Declaration of a Symbol .

IntelliJ IDEA can also provide code completion for HTML tags and component names that you have defined inside methods in JavaScript or inside other components.

Completion also works for imported components with ES6 style syntax. You can navigate from a component name to its definition with `Ctrl+B` or see a definition in a popup with `Ctrl+Shift+I`.

In JSX tags, IntelliJ IDEA provides coding assistance for ReactJS-specific attributes such as `className` or `classID`. Moreover, for class names you can autocomplete classes defined in the project's CSS files.

Code completion is also provided for JavaScript expressions inside curly braces. Code completion applies to all methods and functions that you have defined.

IntelliJ IDEA supports Emmet in JSX code thus enabling some ReactJS-specific twists. For example, the abbreviation `div.my-class` would expand in JSX to `<div className="my-class"></div>` and not to `<div class="my-class"></div>` as it would in HTML.

To get coding assistance with JSX or JavaScript:

1. Make sure you have the `react.js` file anywhere under the project root folder, possibly copy it from the downloaded ReactJS Starter Kit after extraction. Alternatively, configure `react.js` as an external JavaScript library, see Configuring JavaScript Libraries .
2. Do one of the following:
   - Create a `.jsx` file to write your code in. IntelliJ IDEA recognizes this file type and provides coding assistance in it.
   - To use the same coding assistance in `.js` files, change the language level to JSX Harmony: Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click JavaScript under Languages&Frameworks. On the JavaScript page that opens, choose JSX Harmony from the JavaScript Language Version drop-down list.

Enhancing Code Completion with Typed Parameter Information

To see information on parameters that you can use in a method, configure a TypeScript definition file react.d.ts as a JavaScript library and add it to your project:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Libraries under JavaScript.
2. On the JavaScript. Libraries page that opens, click Download on the toolbar.
3. In the Download Library dialog box that opens, select react from the list of stubs, and click Download and Install. You return to the JavaScript. Libraries page where the library is added to the list as Global and is already activated.
4. Copy the `react.d.ts` file to your project.

Using ReactJS with Typescript

IntelliJ IDEA recognizes and supports ReactJS syntax in TypeScript context (TSX) only in `.tsx` files. To use ReactJS with TypeScript code:

1. Add a TypeScript definition file react.d.ts to your project as described above in the section Enhancing Code Completion with Typed Parameter Information or add the `react.js` to your project.
2. Write your code in `.tsx` files.

JavaServer Faces (JSF)

In this section:

- JavaServer Faces (JSF)
  - JSF support
- Preparing for JSF Application Development
- Defining Navigation Rules

JSF support

IntelliJ IDEA's support for JavaServer Faces (JSF) includes:

- JSF-aware coding assistance with code completion .
- JSF code formatting and folding as well as syntax and error highlighting.
- Graphical editor for defining page navigation rules.
- JSF 2.0 support, in particular:
  - Annotated Managed Beans support with code completion, rename refactoring, usage search, go to declaration, and more. Possibility to reference managed beans directly from Java code.
  - Improved template support (completion, validation and navigation for the `name` attribute of the `ui:define` tag).
  - Support for ActionSources and ValueHolders.
  - Support for the `targets` attribute of composite components.
  - Extended support for EL in composite components implementation, and more.
- Dedicated JSF tool window that aggregates all JSF-related configurations in a single place, and lets you easily analyze and navigate to both annotated and xml elements.
- `@ResourceDependencies` and `@ResourceDependency` annotations support.
- Simplified navigation rules support including completion for neighbor pages and absolute paths, usage search, and rename refactoring.
- Resource Handlers support: code completion, syntax and error highlighting, usage search and refactorings.
- Composite components support: code completion, refactorings, usage search and more.
- Code completion, error highlighting, and basic refactorings for the most popular JSF component libraries, including:
  - RichFaces
  - ICEfaces
  - PrimeFaces
  - OpenFaces

Preparing for JSF Application Development

To prepare for JSF application development, you should:

- Make sure that the Java EE: Java Server Faces plugin is enabled. This plugin is bundled with the IDE and enabled by default. However, you may have disabled it for some reason by now.
- Create a project or module with JSF support enabled, or enable JSF support in an existing module.

On this page:

- Making sure that the Java Server Faces plugin is enabled
- Enabling JSF support when creating a project or module
- Enabling JSF support for an existing module

Making sure that the Java Server Faces plugin is enabled

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, select Plugins.
3. In the right-hand part of the dialog, on the Plugins page , type `faces` in the search box. As a result, only the plugins whose names and descriptions contain `faces` are shown in the list of plugins.
4. If the check box to the right of Java EE: Java Server Faces is not selected, select it.
5. Click OK in the Settings dialog.
6. If suggested, restart IntelliJ IDEA.

Enabling JSF support when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java Enterprise . In the right-hand part of the page, specify the JDK to be used and select the Java EE version to be supported.
3. If, at this step, you are ready to specify the application server you are going to deploy your application to (e.g. to test the application behavior), do so. This will result in the corresponding server-specific run/debug configuration for your module generated automatically. Otherwise, to be able to run your application, you will have to create the run/debug configuration yourself.
   Select the server from the list or click New and select the server of interest. Then, specify the server settings:
   - For a server installed locally, specify the path to the server installation directory. (Click … to select the directory in the corresponding dialog .)
   - For a hosted server (Cloud Foundry or CloudBees), specify your user account details.
4. Under Additional Libraries and Frameworks, select the Web Application check box.
   Select the version of the Servlet specification to be supported from the Versions list.

   If you want the deployment descriptor `web.xml` file to be created, select the Create web.xml check box.
5. Select the JSF check box.
   If you want the configuration file `faces-config.xml` to be created, select the Create faces-config.xml check box.

   Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download and use a JSF implementation library (Mojarra).
     To do that, under Libraries, select Download.

     Now, to view or modify the associated options, click Configure, and in the Downloading Options dialog that opens:
     - Select the library version.
     - Specify the library name.
     - Select the library level (global, project, or module).
     - Under Files to download, select which of the files you want to download.
     - Under Copy downloaded files to, specify the path to the destination folder. If you want to change the default path, click … and specify the folder location in the dialog that opens .
   - Use a JSF library IntelliJ IDEA is already aware of.
     To do that, click Use library and select the required library from the list.

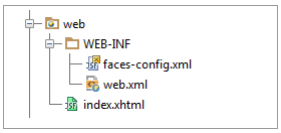
     If necessary, configure the library settings (for example, change its name). This is done in the Edit Library dialog which you can open by clicking Configure.
   - Create a new library using the appropriate JAR files available on your computer.
     To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (For multiple selection, keep the `Ctrl` key pressed.)

     If necessary, configure the new library (for example, change its name or level). To do that, click Configure and specify the required settings in the Create Library dialog.
   - Postpone setting up the library until a later time. In this case, select Set up library later.
6. If you are going to use a JSF component library or libraries (e.g. PrimeFaces, RichFaces, etc.), select the corresponding check box or check boxes and specify the associated options. The procedure is similar to that for the JSF implementation library.
   Click Next.

7. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

As a result, your new module will contain:

- The `web` and `WEB-INF` directories.
- The file `index.xhtml` in the `web` directory. With minor modifications, you can use this file as a starting page of your application.
- In the `WEB-INF` directory, if specified:
  - `web.xml`, the Web application deployment descriptor.
  - `faces-config.xml`, the JSF configuration file.



- If specified, the JSF library or libraries included in the module dependencies .
- An artifact specification for you module.
- If you have specified the server, a run/debug configuration for running your application in the context of that server.

Enabling JSF support for an existing module

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the module and select Add Framework Support.
3. In the left-hand pane of the Add Frameworks Support dialog, select the Web Application check box.
   In the right-hand part of the dialog, select the version of the Servlet specification to be supported from the Versions list.

   If you want the deployment descriptor `web.xml` file to be created, select the Create web.xml check box.
4. Select the JSF check box.
   If you want the configuration file `faces-config.xml` to be created, select the Create faces-config.xml check box.

   Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download and use a JSF implementation library (Mojarra).
     To do that, under Libraries, select Download.

     Now, to view or modify the associated options, click Configure, and in the Downloading Options dialog that opens:
     - Select the library version.
     - Specify the library name.
     - Select the library level (global, project, or module).
     - Under Files to download, select which of the files you want to download.
     - Under Copy downloaded files to, specify the path to the destination folder. If you want to change the default path, click and specify the folder location in the dialog that opens .
   - Use a JSF library IntelliJ IDEA is already aware of.
     To do that, click Use library and select the required library from the list.

     If necessary, configure the library settings (for example, change its name). This is done in the Edit Library dialog which you can open by clicking Configure.
   - Create a new library using the appropriate JAR files available on your computer.
     To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (For multiple selection, keep the `Ctrl` key pressed.)

     If necessary, configure the new library (for example, change its name or level). To do that, click Configure and specify the required settings in the Create Library dialog.
   - Postpone setting up the library until a later time. In this case, select Set up library later.
5. If you are going to use a JSF component library or libraries (e.g. PrimeFaces, RichFaces, etc.), select the corresponding check box or check boxes and specify the associated options. The procedure is similar to that for the JSF implementation library.
6. Click OK in the Add Frameworks Support dialog.

Defining Navigation Rules

Navigation diagrams provide additional support for managing the rules of navigation between the pages of your application. In certain cases, they are a useful alternative to editing the contents of `faces-config.xml`.

With `faces-config.xml` open in the editor, use the Navigation tab to view and draw the diagram. Your drawing is automatically synchronized with the contents of `faces-config.xml` (and vice versa).

For detailed information about the available controls and context menu commands, see Diagram Toolbar and Context Menu.

Using the navigation diagram to create the navigation rules

1. Open `faces-config.xml` in the editor.
2. Click the Navigation tab.
3. Drag the required pages (one by one) from the Project tool window onto the Navigation tab.
4. Use the mouse to draw the navigation lines between the pages. As a result, the corresponding navigation rules are created. (Switch onto the Text tab to see those rules.)

JPA and Hibernate

- Java Persistence API (JPA)
- Hibernate
- Working with the Persistence Tool Window

Java Persistence API (JPA)

- Overview of JPA Support
- Enabling JPA Support
- Working with the Persistence Tool Window
- Working with the JPA Console

Overview of JPA support

For working with Java Persistence API (JPA), IntelliJ IDEA provides:

- The Java EE: EJB, JPA, Servlets plugin . This plugin is bundled with the IDE and enabled by default. See Making sure that the Java EE: EJB, JPA, Servlets plugin is enabled.
- An ability to turn on JPA support for a module . You can do that when creating a project or module, or for an existing project or module. See Enabling JPA Support .
- A JPA facet for managing JPA configuration (`persistence.xml`) and object/relational mapping (`orm.xml`) files. See Hibernate and JPA Facet Pages .
- The Persistence tool window that shows your JPA project items and lets you create new configuration files and persistent classes, navigate to related source code in the editor, open diagrams and consoles, and more. See Working with the Persistence Tool Window .
- An ability to generate managed entity classes and object/relational mappings for them by importing a database schema, an EJB deployment descriptor file `ejb-jar.xml` or a Hibernate object/relational mapping file `.hbm.xml` . See Generating managed entity classes and O/R mappings.
- Entity-relationship (ER) diagrams.
  The ER diagrams are accessed from the Persistence tool window (see Opening entity-relationship diagrams) and provide about the same set of functions as the tool window.

  Those functions are accessed as context menu commands. The context menus are different for the entity classes and the main diagram area (which corresponds to a <persistence-unit>).
- A JPA console that lets you write and run JPQL queries, and analyze the query results. See Working with the JPA Console .
- Code completion in Java code including JPA annotations and their attributes, and also in JPA configuration and O/R mapping XML files.
- JPA-specific code inspections .
- Navigation markers in the left margin of the editor e.g. for jumping from entity classes to corresponding fragments in mapping files.

Enabling JPA Support

- Overview
- Making sure that the Java EE: EJB, JPA, Servlets plugin is enabled
- Enabling JPA support when creating a project or module
- Enabling JPA support for an existing module

Overview

To be able to use JPA support, you should:

- Make sure that the Java EE: EJB, JPA, Servlets plugin is enabled. (This plugin is bundled with the IDE and enabled by default.)
- Enable JPA support at a module level. You can do that when creating a new project or module. You can also enable JPA support for an existing module. In all such cases, IntelliJ IDEA will (some of the following will be offered as options):
  - Create a JPA configuration file `persistence.xml`.
  - Download the library files that implement the framework and add them to the dependencies of the corresponding module.
  - Generate entity classes and object/relational mappings for your database tables (if you have an appropriate data source available).
  - Create a JPA facet. You'll be able to use that facet for specifying the default configuration and object-relational mapping files, and JPA implementation provider.
  - Make the Persistence tool window available.

Making sure that the Java EE: EJB, JPA, Servlets plugin is enabled

1. Open the Settings / Preferences dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, select Plugins.
3. In the right-hand part of the dialog, on the Plugins page, type `jpa` in the search box. As a result, only the plugins whose names and descriptions contain `jpa` are shown.
4. If the check box to the right of Java EE: EJB, JPA, Servlets is not selected, select it.
5. Click OK.
6. If suggested, restart IntelliJ IDEA.

Enabling JPA support when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java. In the right-hand part of the page, specify the JDK that you are going to use.
3. Under Additional Libraries and Frameworks, select the JavaEE Persistence check box.
4. Select the version of `persistence.xml` to be created. (`persistence.xml` is a JPA configuration file.)
   If necessary, select the Java persistence provider, i.e. the JPA implementation provider. (This setting, initially, will only affect the `<provider>` element in `persistence.xml`.)

   If there is a database defined in IntelliJ IDEA as a data source, you can select to import the database schema. This will result in creating the necessary JPA entity classes and object/relational mappings for them.
5. Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download JPA implementation files and arrange those files in a library.
     To do that, under Libraries, select Download. Optionally, click Configure to edit the library settings. (The Downloading Options dialog will open.)
   - Use a library that is already defined IntelliJ IDEA.
     To do that, click Use library and select the required library from the list. Optionally, click Configure to edit the library settings. (The Edit Library dialog will open.)
   - Create a new library using the appropriate JAR files available on your computer.
     To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens. (For multiple selection, keep the `Ctrl` key pressed.)
     Optionally, click Configure to edit the library settings. (The Create Library dialog will open.)
   - Postpone setting up the library until a later time. In this case, select Set up library later.

   Click Next.
6. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location.
   Click Finish.

If you selected to import a database schema, the Import Database Schema dialog opens, and you can specify the entity classes to be generated and associated settings.

Enabling JPA support for an existing module

1. In the Project tool window, right-click the necessary module and select Add Framework Support.
2. In the left-hand pane of the Add Frameworks Support dialog that opens, select the JavaEE Persistence check box.
3. Select the version of `persistence.xml` to be created. (`persistence.xml` is a JPA configuration file.)
   If necessary, select the Java persistence provider, i.e. the JPA implementation provider. (This setting, initially, will only affect the `<provider>` element in `persistence.xml`.)

   If there is a database defined in IntelliJ IDEA as a data source, you can select to import the database schema. This will result in creating the necessary JPA entity classes and object/relational mappings for them.
4. Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download JPA implementation files and arrange those files in a library.
     To do that, under Libraries, select Download. Optionally, click Configure to edit the library settings. (The Downloading Options dialog will open.)
   - Use a library that is already defined IntelliJ IDEA.
     To do that, click Use library and select the required library from the list. Optionally, click Configure to edit the library settings. (The Edit Library dialog will open.)

- Create a new library using the appropriate JAR files available on your computer.
  To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (For multiple selection, keep the `Ctrl` key pressed.)
  Optionally, click Configure to edit the library settings. (The Create Library dialog will open.)
- Postpone setting up the library until a later time. In this case, select Set up library later.

5. Click OK.

If you selected to import a database schema, the Import Database Schema dialog opens, and you can specify the entity classes to be generated and associated settings.

Working with the JPA console

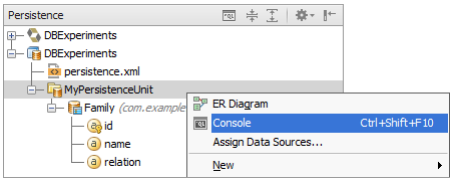
Use the JPA console to write and run JPQL queries, analyze the query results, and also to perform other, associated tasks.

- Opening the JPA console
- Viewing and modifying console settings
- Composing JPQL queries
- Navigating to the declaration of a class or field
- Running a query
- Running parameterized queries
- Running auto-memorized queries
- Terminating query execution
- Generating SQL statements and DDL SQL scripts
- Hiding or showing the toolbar
- Pinning the Result tab
- Switching between subsets of rows
- Making all rows visible simultaneously
- Navigating to a specified row
- Sorting data
- Reordering columns
- Hiding and showing columns
- Restoring the initial table view
- Using the Structure view to sort data, and hide and show columns
- Copying table data to the clipboard or saving them in a file
- Changing data output format
- Saving a LOB in a file
- Updating the table view
- Viewing the query
- Closing a console

See also, JPA Console Tool Window .

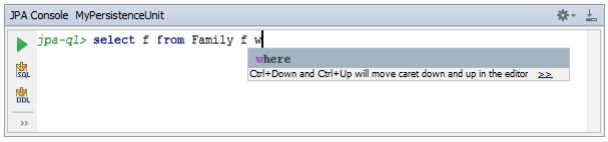
Opening the JPA console

1. Open the Persistence tool window (e.g. View | Tool Windows | Persistence).
2. Expand the JPA facet node.
3. Select the persistence unit for which you want to open the console or any node within that persistence unit.
4. Do one of the following:
   - Click on the title bar.
   - Select Console from the context menu.
   - Press `Ctrl+Shift+F10`.



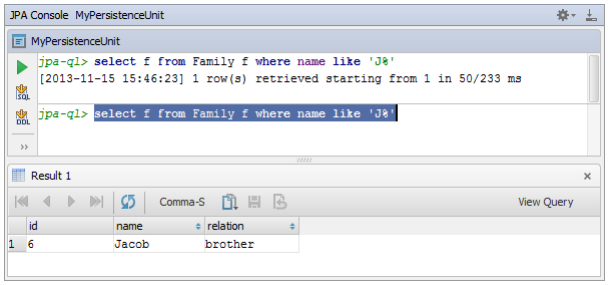
5. If asked to choose the console, select JPA Console.

As a result, the JPA Console tool window opens and the input pane is shown. This is where you compose your JPQL queries.

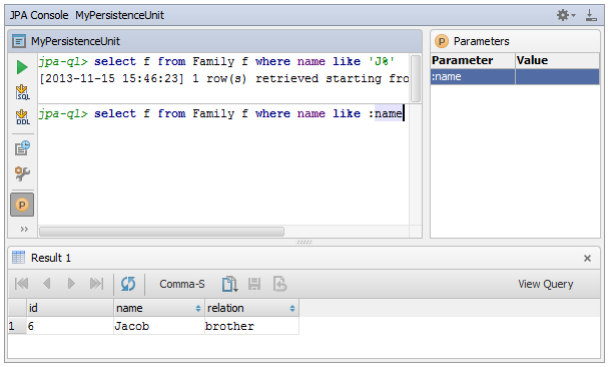


When you run your first query ( ), the output pane opens above the input pane. Basically, this is the log of operations performed in the console.

If your query retrieves data (e.g. `select`), also the Result pane opens showing the retrieved data in table format.



Additionally, you can open the Parameters pane ( ) to manage parameters in your queries.



Viewing and modifying console settings

Before actually starting to use a console, you may want to take a look at the console settings and adjust them to your needs.

– To access these settings, click on the toolbar of the JPA Console tool window. (Alternatively, `Ctrl+Alt+S` | Tools | Database.)

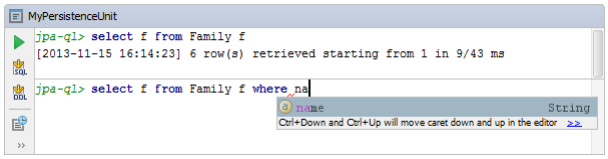
As a result, the Database page of the Settings / Preferences dialog will open. The settings for the JPA console are on the following pages:

– Data Views

– CSV Formats

Composing JPQL queries

When composing your queries in the input pane, use auto-completion and highlighting of JPQL keywords, and object and property names.



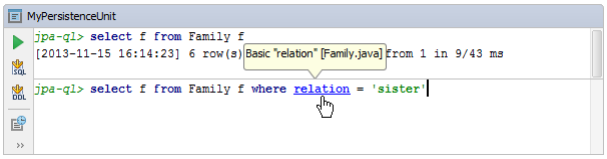
See also, Navigating to the declaration of a class or field.

Navigating to the declaration of a class or field

When composing a query, it's sometimes useful to take a look at the declaration of a class or field for an object or property referenced in the input pane. To navigate to the corresponding declaration, do one of the following:

– Place the cursor within the name of the object or property of interest. Then use `Ctrl+B`. (Alternatively, you can use Navigate | Declaration from the main menu.)

– Press and hold the `Ctrl` key, and point to the name of interest. When the text turns into a hyperlink, click the hyperlink.



As a result, the necessary source file opens in the editor and the cursor is placed within the declaration of the corresponding class or the getter method for the corresponding field.

Running a query

To run the current query, do one of the following:

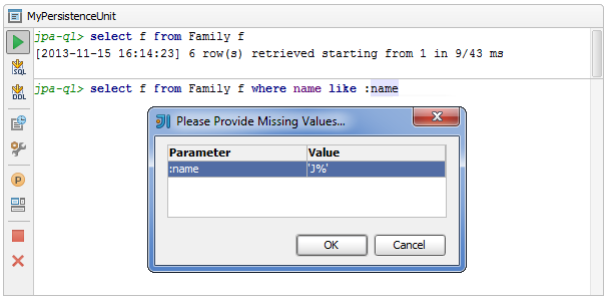
– Click on the toolbar.

– Press `Ctrl+Enter`.

Running parameterized queries

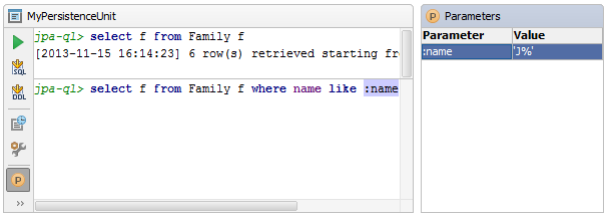
Your queries can contain parameters, however, by the time you run such queries the values of the parameters must be specified. There are the following ways of specifying the parameter values:

– Click on the toolbar or press `Ctrl+Enter` to run the query. In the dialog that opens, specify the parameter values and click OK.



(To start editing a value, switch to the corresponding table cell and start typing. To indicate that you have finished editing a value, press `Enter` or switch to a different cell. To quit the editing mode and restore an initial value, press `Escape`.)

– Alternatively, you can open the Parameters pane (on the toolbar) and specify the corresponding values there. (The values are edited in the same way as in the corresponding dialog.) Then run the query (on the toolbar or `Ctrl+Enter`).



Parameter values can be specified just as text or numbers, or as Groovy expressions that contain object references and method calls. For example, the value for the `date` parameter in the query

```
SELECT o
                    FROM Order o
                    WHERE o.date >
                    :date
```

could be specified as

```
new java.sql.Date(System.currentTimeMillis() - 24*3600*1000)
```

Running auto-memorized queries

As you run JPQL queries in the console, IntelliJ IDEA memorizes them. So, at a later time, you can view the queries you have already run and, if necessary, run them again.

To open the dialog where the auto-memorized queries are shown (the History dialog), do one of the following:

- Click on the toolbar.
- Press `Ctrl+Alt+E`.

There are two panes in the History dialog. The left-hand pane shows the list of the queries that you have run. For "long" queries, only their beginnings are shown. When you select a query in this pane, the overall query is shown in the pane to the right.

You can filter the information: just start typing. As a result, only the queries that contain the typed text will be shown.

You can copy the queries from the History dialog into the input pane of the console. To copy a query, do one of the following:

- Double-click the query to be copied.
- Select the query of interest and press `Enter`.
- Select the query and click OK.

(Once the query is in the input pane, you can run it straight away.)

You can delete unnecessary memorized queries. To delete a query, select the query in the History dialog and press `Delete`.

Terminating query execution

To terminate execution of the current query, do one of the following:

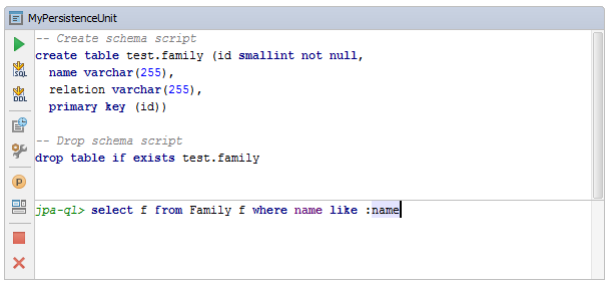
- Click on the toolbar.
- Press `Ctrl+F2`.

Generating SQL statements and DDL SQL scripts

You can generate SQL statements for your JPQL queries and DDL SQL scripts for your persistence unit:

- To generate an SQL equivalent of the current query, do one of the following:
  - Click on the toolbar.
  - Press `Ctrl+Shift+Enter`.
- To generate DDL SQL statements (`CREATE TABLE` , `ALTER TABLE` and `DROP TABLE`) for all the objects (classes) associated with the corresponding persistence unit, do one of the following:
  - Click on the toolbar.
  - Press `Ctrl+Shift+Alt+Enter`.

The generated SQL statements are shown in the output pane.



Hiding or showing the toolbar

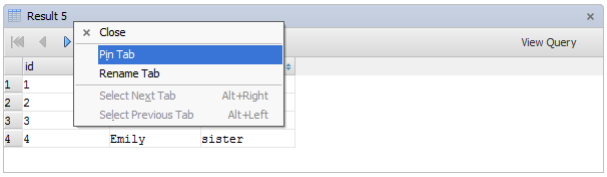
To hide or show the toolbar of the Result pane:

– Click on the title bar of the JPA Console tool window and click Show Toolbar.

Pinning the Result tab

If one and the same tab is used to show your query results, and you get the result that you want to keep, you can pin the tab to the tool window. To do that:

– Right-click the tab and select Pin Tab.



See also, Show query results in new tab .

Switching between subsets of rows

If only a subset of the rows that satisfy the query is currently shown, to switch between the subsets, use:

– First Page

– Previous Page (`Ctrl+Alt+Up`)

– Next Page (`Ctrl+Alt+Down`)

– Last Page

See also, Making all rows visible simultaneously.

Making all rows visible simultaneously

If you want all the rows that satisfy the query to be shown simultaneously:

1. Click on the toolbar of the JPA Console tool window.
2. Switch to the Database | Data Views page, specify `0` in the Result set page size field, and click OK.
3. Click or press `Ctrl+F5` to refresh the table view.

See also, Updating the table view and Result set page size .

Navigating to a specified row

To switch to a row with a specified number:

1. Do one of the following:
   – Press `Ctrl+G` .
   – Right-click the table and select Go To | Row from the context menu.
   – Select Navigate | Row from the main menu.
2. In the dialog that opens, specify the row number and click OK.

Sorting data

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: name . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance: name 1. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

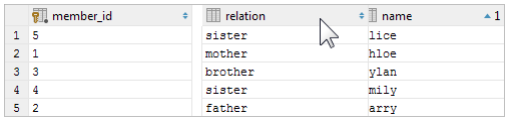
When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: name 1.

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: name .

See also, Restoring the initial table view and Using the Structure view to sort data, and hide and show columns.

Reordering columns

To reorder columns, use drag-and-drop for the corresponding cells in the header row.



See also, Restoring the initial table view.

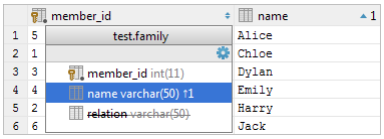
Hiding and showing columns

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
   – Right-click any of the cells in the header row and select Column List.
   – Press `Ctrl+F12` .

   In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

See also, Restoring the initial table view and Using the Structure view to sort data, and hide and show columns.

Restoring the initial table view

Click on the toolbar to restore the initial table view after reordering or hiding the columns, or sorting the data. As a result, the data, generally, becomes unsorted, the columns appear in the order they are defined in the corresponding query, and all the columns are shown.

Using the Structure view to sort data, and hide and show columns

When working with the Result pane, the table structure view is available as the corresponding popup.

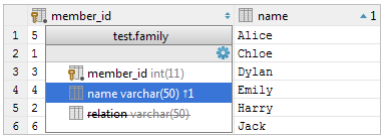
The structure view shows the list of all the columns and lets you sort the data as well as hide and show the columns.

To open the structure popup, do one of the following:

- Right-click a cell in the table header row and select Column List.
- Press `Ctrl+F12`.

In the popup, select the column of interest and do one of the following:

- To sort the data by this column in the ascending order, press `Shift+Alt+Up`.
- To sort the data in the descending order, press `Shift+Alt+Down`.
- To cancel sorting by this column, press `Ctrl+Shift+Alt+Backspace`.
- To hide the column (or show a hidden column), press `Space`. (The names of hidden columns are shown struck through.)



The shortcuts for sorting table data (`Shift+Alt+Up`, `Shift+Alt+Down` and `Ctrl+Shift+Alt+Backspace`) can be used in the Result pane without opening the structure view.

See also, Sorting data, Hiding and showing columns and Restoring the initial table view.

Copying table data to the clipboard or saving them in a file

1. Do one of the following:
   - Click on the toolbar and select Copy Data To Clipboard or Save Data To File.
   - Right-click the table and select Copy Data To Clipboard or Save Data To File from the context menu.
2. If you are saving the data in a file, select the output format (the extractor), and then specify the file name and location.

If only a subset of rows is currently shown, all the rows that satisfy the corresponding query are copied to the clipboard or saved in a file anyway.

See also, Changing data output format.

Changing data output format

When you copy table data to the clipboard (the Copy and Dump Data | To Clipboard commands), unless you change the format, the data are output as Tab-Separated Values (TSV).

To change the output format or to define a new format for the output:

1. Do one of the following:
   - Click Tab S...(TSV) on the toolbar.
   - Right-click the table and point to Data Extractor: <current_format>.
2. Do one of the following:
   - To select a different output format, click the name of the format.
   - To define a new delimiter-separated values format, or to view or modify the settings for an existing one, click Configure CSV Formats. As a result, the CSV Formats Dialog will open.

Saving a LOB in a file

If a cell contains a binary large object (a.k.a. BLOB or LOB), you can save such a LOB in a file.

1. Right-click the cell that contains the LOB of interest and select Save LOB To File.
2. In the dialog that opens, specify the name and location of the destination file and click OK.

Updating the table view

To refresh the table view, do one of the following:

- Click on the toolbar.
- Right-click the table and select Reload Page from the context menu.
- Press `Ctrl+F5`.

Use this function to:

- Synchronize the data shown with the actual contents of the database.
- Apply the Result set page size setting after its change.

Viewing the query

To see the query that was used to generate the table:

- Click View Query on the toolbar.
  If necessary, you can select the query text and copy it to the clipboard (`Ctrl+C`).

To close the pane where the query is shown, press `Escape` .

Closing a console

To close a console, do one of the following:

- Click on the toolbar.
- Press `Ctrl+Shift+F4` .

Hibernate

- Overview of Hibernate Support
- Enabling Hibernate Support
- Working with the Persistence Tool Window
- Working with the Hibernate Console

Overview of Hibernate support

For working with Hibernate , IntelliJ IDEA provides:

- The Hibernate Support plugin . This plugin is bundled with the IDE and enabled by default. See Making sure that the Hibernate Support plugin is enabled.
- An ability to turn on Hibernate support for a module . You can do that when creating a project or module, or for an existing project or module. See Enabling Hibernate Support .
- A Hibernate facet for managing Hibernate configuration files (`hibenate.cfg.xml`). See Hibernate and JPA Facet Pages .
- The Persistence tool window that shows your Hibernate project items and lets you create new configuration files and persistent classes, navigate to related source code in the editor, open diagrams and consoles, and more. See Working with the Persistence Tool Window .
- An ability to generate managed entity classes and object/relational mappings for them by importing a database schema or an EJB deployment descriptor file `ejb-jar.xml` . See Generating managed entity classes and O/R mappings.
- Entity-relationship (ER) diagrams.
  The ER diagrams are accessed from the Persistence tool window (see Opening entity-relationship diagrams) and provide about the same set of functions as the tool window.

  Those functions are accessed as context menu commands. The context menus are different for the entity classes and the main diagram area (which corresponds to a <session-factory>).
- A Hibernate console that lets you write and run HQL queries, and analyze the query results. See Working with the Hibernate Console .
- Code completion in Java code including Hibernate annotations and their attributes, and also in Hibernate configuration and O/R mapping XML files.
- Hibernate-specific code inspections .
- Navigation markers in the left margin of the editor e.g. for jumping from entity classes to corresponding fragments in mapping files.

Enabling Hibernate Support

- Overview
- Making sure that the Hibernate Support plugin is enabled
- Enabling Hibernate support when creating a project or module
- Enabling Hibernate support for an existing module

Overview

To be able to use Hibernate support, you should:

- Make sure that the Hibernate Support plugin is enabled. (This plugin is bundled with the IDE and enabled by default.)
- Enable Hibernate support at a module level. You can do that when creating a new project or module. You can also enable Hibernate support for an existing module. In all such cases, IntelliJ IDEA will (some of the following will be offered as options):
  - Create a Hibernate configuration file `hibernate.cfg.xml` and a class with a `main()` method that outputs the records for your managed entities.
  - Download the library files that implement the framework and add them to the dependencies of the corresponding module.
  - Generate entity classes and object/relational mappings for your database tables (if you have an appropriate data source available).
  - Create a Hibernate facet . You'll be able to use that facet for managing your configuration `hibernate.cfg.xml` files.
  - Make the Persistence tool window available.

Making sure that the Hibernate Support plugin is enabled

1. Open the Settings / Preferences dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, select Plugins.
3. In the right-hand part of the dialog, on the Plugins page , type `hib` in the search box. As a result, only the plugins whose names and descriptions contain `hib` are shown.
4. If the check box to the right of Hibernate Support is not selected, select it.
5. Click OK.
6. If suggested, restart IntelliJ IDEA.

Enabling Hibernate support when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java . In the right-hand part of the page, specify the JDK that you are going to use.
3. Under Additional Libraries and Frameworks, select the Hibernate check box.
4. If necessary, select to create a configuration file (`hibernate.cfg.xml`) and a main class (a class with a `main()` method that outputs the records for your managed entities).
   If there is a database defined in IntelliJ IDEA as a data source, you can select to import the database schema. This will result in creating the necessary entity classes and object/relational mappings for them.
5. Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download Hibernate implementation files and arrange those files in a library .
     To do that, under Libraries, select Download. Optionally, click Configure to edit the library settings. (The Downloading Options dialog will open.)
   - Use a library that is already defined IntelliJ IDEA.
     To do that, click Use library and select the required library from the list. Optionally, click Configure to edit the library settings. (The Edit Library dialog will open.)
   - Create a new library using the appropriate JAR files available on your computer.
     To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (For multiple selection, keep the `Ctrl` key pressed.)
     Optionally, click Configure to edit the library settings. (The Create Library dialog will open.)
   - Postpone setting up the library until a later time. In this case, select Set up library later.

   Click Next.
6. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

If you selected to import a database schema, the Import Database Schema dialog opens, and you can specify the entity classes to be generated and associated settings.

Enabling Hibernate support for an existing module

1. In the Project tool window , right-click the necessary module and select Add Framework Support.
2. In the left-hand pane of the Add Frameworks Support dialog that opens, select the Hibernate check box.
3. If necessary, select to create a configuration file (`hibernate.cfg.xml`) and a main class (a class with a `main()` method that outputs the records for your managed entities).
   If there is a database defined in IntelliJ IDEA as a data source, you can select to import the database schema. This will result in creating the necessary entity classes and object/relational mappings for them.
4. Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download Hibernate implementation files and arrange those files in a library .
     To do that, under Libraries, select Download. Optionally, click Configure to edit the library settings. (The Downloading Options dialog will open.)
   - Use a library that is already defined IntelliJ IDEA.
     To do that, click Use library and select the required library from the list. Optionally, click Configure to edit the library settings. (The Edit Library dialog will open.)
   - Create a new library using the appropriate JAR files available on your computer.
     To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (For multiple selection, keep the `Ctrl` key pressed.)
     Optionally, click Configure to edit the library settings. (The Create Library dialog will open.)

- Postpone setting up the library until a later time. In this case, select Set up library later.

5. Click OK.

If you selected to import a database schema, the Import Database Schema dialog opens, and you can specify the entity classes to be generated and associated settings.

Working with the Hibernate console

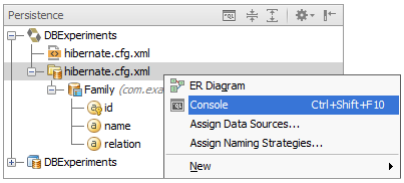
Use the Hibernate console to write and run HQL queries, analyze the query results, and also to perform other, associated tasks.

- Opening the Hibernate console
- Viewing and modifying console settings
- Composing HQL queries
- Navigating to the declaration of a class or field
- Running a query
- Running parameterized queries
- Running auto-memorized queries
- Terminating query execution
- Generating SQL statements and DDL SQL scripts
- Hiding or showing the toolbar
- Pinning the Result tab
- Switching between subsets of rows
- Making all rows visible simultaneously
- Navigating to a specified row
- Sorting data
- Reordering columns
- Hiding and showing columns
- Restoring the initial table view
- Using the Structure view to sort data, and hide and show columns
- Copying table data to the clipboard or saving them in a file
- Changing data output format
- Saving a LOB in a file
- Updating the table view
- Viewing the query
- Closing a console

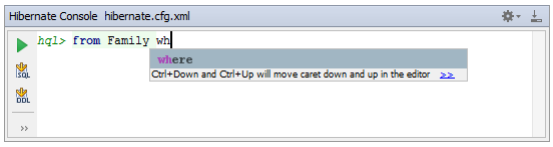
See also, Hibernate Console Tool Window .

Opening the Hibernate console

1. Open the Persistence tool window (e.g. View | Tool Windows | Persistence).
2. Expand the Hibernate facet node.
3. Select the session factory for which you want to open the console or any node within that session factory.
4. Do one of the following:
   - Click on the title bar.
   - Select Console from the context menu.
   - Press `Ctrl+Shift+F10`.

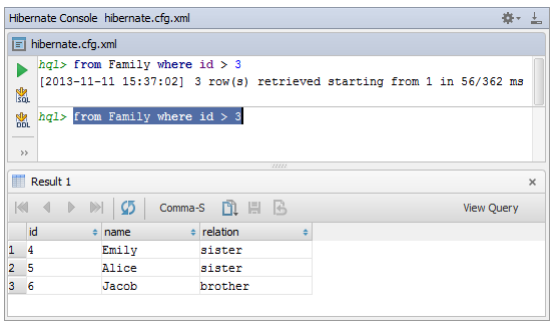


As a result, the Hibernate Console tool window opens and the input pane is shown. This is where you compose your HQL queries.

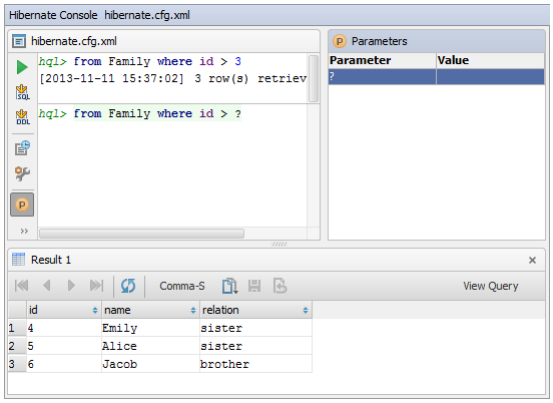


When you run your first query ( ), the output pane opens above the input pane. Basically, this is the log of operations performed in the console.

If your query retrieves data (e.g. `from` , `select`), also the Result pane opens showing the retrieved data in table format.



Additionally, you can open the Parameters pane ( ) to manage parameters in your queries.



Viewing and modifying console settings

Before actually starting to use a console, you may want to take a look at the console settings and adjust them to your needs.

– To access these settings, click on the toolbar of the Hibernate Console tool window. (Alternatively, `Ctrl+Alt+S` | Tools | Database.)

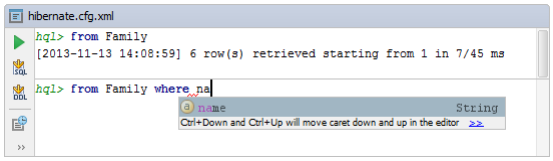
As a result, the Database page of the Settings / Preferences dialog will open. The settings for the Hibernate console are on the following pages:

– Data Views

– CSV Formats

Composing HQL queries

When composing your queries in the input pane, use auto-completion and highlighting of HQL keywords, and object and property names.



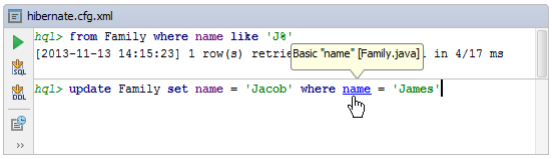
See also, Navigating to the declaration of a class or field.

Navigating to the declaration of a class or field

When composing a query, it's sometimes useful to take a look at the declaration of a class or field for an object or property referenced in the input pane. To navigate to the corresponding declaration, do one of the following:

– Place the cursor within the name of the object or property of interest. Then use `Ctrl+B`. (Alternatively, you can use Navigate | Declaration from the main menu.)

– Press and hold the `Ctrl` key, and point to the name of interest. When the text turns into a hyperlink, click the hyperlink.



As a result, the necessary source file opens in the editor and the cursor is placed within the declaration of the corresponding class or the getter method for the corresponding field.

Running a query

To run the current query, do one of the following:

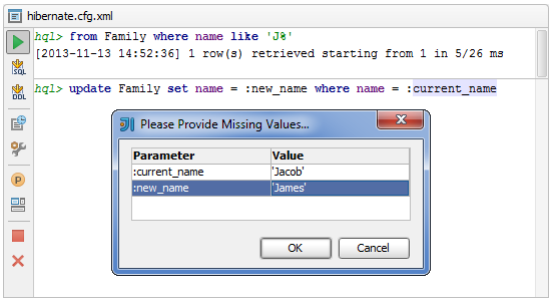
– Click on the toolbar.

– Press `Ctrl+Enter`.

Running parameterized queries

Your queries can contain parameters, however, by the time you run such queries the values of the parameters must be specified. There are the following ways of specifying the parameter values:

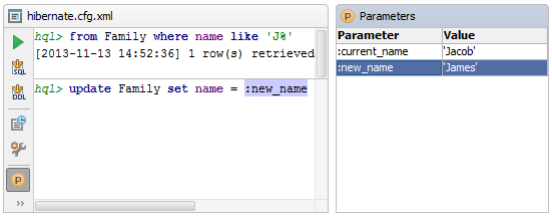
– Click on the toolbar or press `Ctrl+Enter` to run the query. In the dialog that opens, specify the parameter values and click OK.



(To start editing a value, switch to the corresponding table cell and start typing. To indicate that you have finished editing a value, press `Enter` or switch to a different cell. To quit the editing mode and restore an initial value, press `Escape`.)

– Alternatively, you can open the Parameters pane (on the toolbar) and specify the corresponding values there. (The values are edited in the same way as in the

corresponding dialog.) Then run the query (on the toolbar or `Ctrl+Enter`).



Parameter values can be specified just as text or numbers, or as Groovy expressions that contain object references and method calls. For example, the value for the `date` parameter in the query

```
SELECT o
                    FROM Order o
                    WHERE o.date >
                    :date
```

could be specified as

```
new java.sql.Date(System.currentTimeMillis() - 24*3600*1000)
```

Running auto-memorized queries

As you run HQL queries in the console, IntelliJ IDEA memorizes them. So, at a later time, you can view the queries you have already run and, if necessary, run them again.

To open the dialog where the auto-memorized queries are shown (the History dialog), do one of the following:

– Click on the toolbar.
– Press `Ctrl+Alt+E`.

There are two panes in the History dialog. The left-hand pane shows the list of the queries that you have run. For "long" queries, only their beginnings are shown. When you select a query in this pane, the overall query is shown in the pane to the right.

You can filter the information: just start typing. As a result, only the queries that contain the typed text will be shown.

You can copy the queries from the History dialog into the input pane of the console. To copy a query, do one of the following:

– Double-click the query to be copied.
– Select the query of interest and press `Enter`.
– Select the query and click OK.

(Once the query is in the input pane, you can run it straight away.)

You can delete unnecessary memorized queries. To delete a query, select the query in the History dialog and press `Delete`.

Terminating query execution

To terminate execution of the current query, do one of the following:

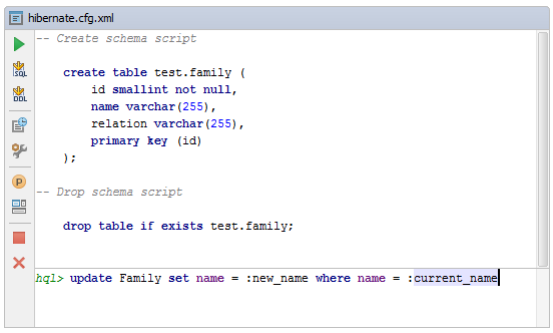
– Click on the toolbar.
– Press `Ctrl+F2`.

Generating SQL statements and DDL SQL scripts

You can generate SQL statements for your HQL queries and DDL SQL scripts for your session factory:

– To generate an SQL equivalent of the current query, do one of the following:
  – Click on the toolbar.
  – Press `Ctrl+Shift+Enter`.
– To generate DDL SQL statements (`CREATE TABLE` , `ALTER TABLE` and `DROP TABLE`) for all the objects (classes) associated with the corresponding session factory, do one of the following:
  – Click on the toolbar.
  – Press `Ctrl+Shift+Alt+Enter`.

The generated SQL statements are shown in the output pane.



Hiding or showing the toolbar

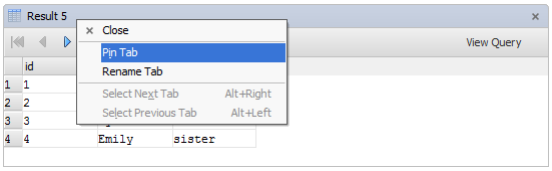
To hide or show the toolbar of the Result pane:

– Click on the title bar of the Hibernate Console tool window and click Show Toolbar.

Pinning the Result tab

If one and the same tab is used to show your query results, and you get the result that you want to keep, you can pin the tab to the tool window. To do that:

– Right-click the tab and select Pin Tab.



See also, Show query results in new tab .

Switching between subsets of rows

If only a subset of the rows that satisfy the query is currently shown, to switch between the subsets, use:

– First Page

– Previous Page (`Ctrl+Alt+Up`)

– Next Page (`Ctrl+Alt+Down`)

– Last Page

See also, Making all rows visible simultaneously.

Making all rows visible simultaneously

If you want all the rows that satisfy the query to be shown simultaneously:

1. Click on the toolbar of the Hibernate Console tool window.
2. Switch to the Database | Data Views page, specify `0` in the Result set page size field, and click OK.
3. Click or press `Ctrl+F5` to refresh the table view.

See also, Updating the table view and Result set page size .

Navigating to a specified row

To switch to a row with a specified number:

1. Do one of the following:
   – Press `Ctrl+G` .
   – Right-click the table and select Go To | Row from the context menu.
   – Select Navigate | Row from the main menu.
2. In the dialog that opens, specify the row number and click OK.

Sorting data

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: name . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance: name 1 . The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

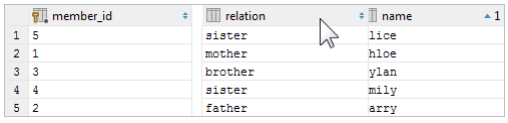
When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: name 1 .

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: name .

See also, Restoring the initial table view and Using the Structure view to sort data, and hide and show columns.

Reordering columns

To reorder columns, use drag-and-drop for the corresponding cells in the header row.



See also, Restoring the initial table view.

Hiding and showing columns

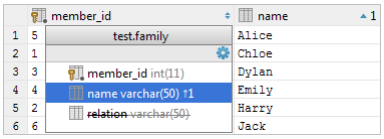
To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
   – Right-click any of the cells in the header row and select Column List.

– Press `Ctrl+F12`.

In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

See also, Restoring the initial table view and Using the Structure view to sort data, and hide and show columns.

Restoring the initial table view

Click on the toolbar to restore the initial table view after reordering or hiding the columns, or sorting the data. As a result, the data, generally, becomes unsorted, the columns appear in the order they are defined in the corresponding query, and all the columns are shown.

Using the Structure view to sort data, and hide and show columns

When working with the Result pane, the table structure view is available as the corresponding popup.

The structure view shows the list of all the columns and lets you sort the data as well as hide and show the columns.

To open the structure popup, do one of the following:

– Right-click a cell in the table header row and select Column List.

– Press `Ctrl+F12`.

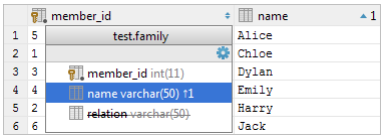
In the popup, select the column of interest and do one of the following:

– To sort the data by this column in the ascending order, press `Shift+Alt+Up`.

– To sort the data in the descending order, press `Shift+Alt+Down`.

– To cancel sorting by this column, press `Ctrl+Shift+Alt+Backspace`.

– To hide the column (or show a hidden column), press `Space`. (The names of hidden columns are shown struck through.)



The shortcuts for sorting table data (`Shift+Alt+Up`, `Shift+Alt+Down` and `Ctrl+Shift+Alt+Backspace`) can be used in the Result pane without opening the structure view.

See also, Sorting data, Hiding and showing columns and Restoring the initial table view.

Copying table data to the clipboard or saving them in a file

1. Do one of the following:
   – Click on the toolbar and select Copy Data To Clipboard or Save Data To File.
   – Right-click the table and select Copy Data To Clipboard or Save Data To File from the context menu.
2. If you are saving the data in a file, select the output format (the extractor), and then specify the file name and location.

If only a subset of rows is currently shown, all the rows that satisfy the corresponding query are copied to the clipboard or saved in a file anyway.

See also, Changing data output format.

Changing data output format

When you copy table data to the clipboard (the Copy and Dump Data | To Clipboard commands), unless you change the format, the data are output as Tab-Separated Values (TSV).

To change the output format or to define a new format for the output:

1. Do one of the following:
   – Click Tab S...(TSV) on the toolbar.
   – Right-click the table and point to Data Extractor: <current_format>.
2. Do one of the following:
   – To select a different output format, click the name of the format.
   – To define a new delimiter-separated values format, or to view or modify the settings for an existing one, click Configure CSV Formats. As a result, the CSV Formats Dialog will open.

Saving a LOB in a file

If a cell contains a binary large object (a.k.a. BLOB or LOB), you can save such a LOB in a file.

1. Right-click the cell that contains the LOB of interest and select Save LOB To File.
2. In the dialog that opens, specify the name and location of the destination file and click OK.

Updating the table view

To refresh the table view, do one of the following:

– Click on the toolbar.

– Click on the toolbar.
– Right-click the table and select Reload Page from the context menu.
– Press `Ctrl+F5`.

Use this function to:

– Synchronize the data shown with the actual contents of the database.
– Apply the Result set page size setting after its change.

Viewing the query

To see the query that was used to generate the table:

– Click View Query on the toolbar.
  If necessary, you can select the query text and copy it to the clipboard (`Ctrl+C`).

To close the pane where the query is shown, press `Escape`.

Closing a console

To close a console, do one of the following:

– Click on the toolbar.
– Press `Ctrl+Shift+F4`.

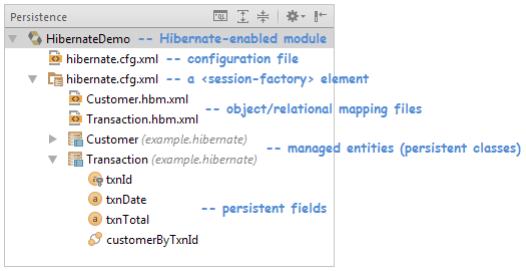
Working with the Persistence tool window

- Overview of the tool window
- Opening the Persistence tool window
- Generating managed entity classes and O/R mappings
- Using the New command
- Opening entity-relationship diagrams
- Associating persistence units and session factories with data sources
- Associating a session factory with a NamingStrategy implementation class (Hibernate)
- Starting a JPQL or an HQL console

See also, Persistence Tool Window .

Overview of the tool window

The Persistence tool window shows your JPA and Hibernate project items, and lets you create configuration files, <persistence-unit> and <session-factory> elements, persistent classes and fields (see Using the New command), navigate to related source code in the editor (F4), open consoles and entity-relationship diagrams, and more.



Opening the Persistence tool window

For the tool window to be available, there must be a JPA- or Hibernate-enabled module in your project, i.e. a module with a JPA or Hibernate facet . See Enabling JPA Support or Enabling Hibernate Support .

To open the tool window, do one of the following:

- Select View | Tool Windows | Persistence
- If the tool window bars are currently shown, click the Persistence button (normally located in the lower-left part of the workspace).
- Point to or in the lower-left corner of the workspace and select Persistence.

Generating managed entity classes and O/R mappings

You can generate managed entity classes and object/relational mappings for them by importing:

- A database schema represented by a data source .
- An EJB facet with an associated deployment descriptor file `ejb-jar.xml` . The facet should be available in the same project. When retrieving the information from `ejb-jar.xml` , only the `<entity>` elements are processed.
- Only for JPA: a Hibernate object/relational mapping file (`.hbm.xml`). The file should be in the same project.

To perform the import:

1. Right-click a module, persistence unit or session factory, point to Generate Persistence Mapping and select one of the following options:
   - By Database Schema
   - By Entity Beans
   - By Hibernate Mappings (this option is not available for Hibernate-enabled modules and session factories)
2. In the dialog that opens , specify the source of import and the output options.

Example: Using a database as a source

1. Right-click a module, persistence unit or session factory, point to Generate Persistence Mapping and select By Database Schema.
2. In the Import Database Schema dialog that opens:
   - General Settings. Select the data source to be used as a source. Specify the destination package for your entity classes, and the prefix and the suffix for the class names.
   - Database Schema Mapping. Select the tables and columns to be mapped, edit the class and field names (the Map As column), and specify the field types (the Mapped Type column).
   - Generation Settings. Specify the target <persistence-unit> or <session-factory>, and select the necessary output options.

Using the New command

The New command lets you create XML configuration files, <persistence-unit> and <session-factory> elements, entity classes, embeddables, mapped superclasses, entity listeners and object-relational mappings, persistent fields in entity classes along with getter and setter method for them, etc.

The New command can be accessed by using the context or the File menu, or the keyboard shortcut `Alt+Insert` . The command options depend on which item is currently selected.

Example: Creating an entity class

1. Right-click a persistence unit or session factory, point to New, and select Entity.
2. In the New Entity dialog that opens, specify:
   - Create class: the name of the class.
   - Destination package: the package in which the class should be created.

- Target destination directory: the destination source root folder (if there is more than one).

3. For JPA, the Select Metadata Targets dialog is shown. Select the files in which you want to add the mapping info for the class:
   - Object/relational mapping XML files: the info will be added as an `<entity>` element within `<entity-mappings>`.
   - The class file itself (.java): the `@Entity` annotation will be added.

   IntelliJ IDEA will also add the info to `persistence.xml` either as a `<class>` or a `<mapping-file>` element.

   For Hibernate, the Select Metadata Targets dialog is not shown. The `@Entity` annotation is added to the class and the `<mapping class="">` element is added to `hibernate.cfg.xml`.

Opening entity-relationship diagrams

To open an entity-relationship diagram, right-click a persistence unit, session factory, entity or field, and select ER Diagram from the context menu.

See also, Diagram Toolbar and Context Menu and General Techniques of Using Diagrams .

Associating persistence units and session factories with data sources

You can associate a <persistence-unit> or <session-factory> element with a data source. If you do so:

- Your source code that references database tables will automatically be validated against the corresponding data source.
- When using the JPA or Hibernate console, you won't need to additionally specify the database connection settings. The corresponding settings configured for the associated DB data source will automatically be used.

If the <persistence-unit> or <session-factory> elements are generated by importing a database schema, the association between the corresponding elements and the data source is set automatically.

To associate a persistence unit or session factory with a data source:

1. Right-click a module, persistence unit, session factory or entity, and select Assign Data Sources.
2. In the dialog that opens, click the necessary Data Source cell and select the data source from the list. (To remove an association with a data source, select <none>.)

Associating a session factory with a NamingStrategy implementation class (Hibernate)

1. Right-click a module, session factory or entity, and select Assign Naming Strategies.
2. In the dialog that opens, click the necessary Naming Strategy cell and select the implementation class from the list.

Starting a JPQL or an HQL console

Right-click a persitence unit or session factory, or any node within it, and select Console.

See also:

- Working with the JPA Console
- Working with the Hibernate Console

Kotlin

In this section:

- Kotlin
  - Prerequisite
  - Kotlin support in IntelliJ IDEA
- Creating Kotlin-JVM Project
- Creating Kotlin-JavaScript Project
- Converting a Java File to Kotlin File
- Mixing Java and Kotlin in One Project

Prerequisite

Before you start working with Kotlin, make sure that the Kotlin plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Kotlin support in IntelliJ IDEA

Kotlin support in IntelliJ IDEA includes:

- Dedicated file type, denoted with icon.
- File and Code Templates for creating Kotlin symbols, which allows producing the Kotlin classes , interfaces , enums and objects .
- Project types: Kotlin-JVM and Kotlin-JavaScript. To learn how to create these projects, refer to the sections Creating Kotlin-JVM Project and Creating Kotlin-JavaScript Project .
- Ability to configure Kotlin modules in a project for JVM, JavaScript, Maven or Gradle.

  For details, refer to this tutorial .
- Kotlin compiler configuration .
- Dedicated Kotlin , Kotlin Script and Kotlin-JavaScript run/debug configurations.

Creating Kotlin-JVM Project

To create a Kotlin-JVM project, follow these steps

1. On the main menu, choose File | New | Project.
2. In the New Project dialog box, select the project type Kotlin-JVM.
3. Click Next.
4. On the next page of the wizard, specify the following:
   - Project name and location.
   - Kotlin runtime library. Select the desired library from the drop-down list, or, if missing, click Create.

   If necessary, click ▸ (More settings)and specify the settings as required.
5. Click Finish.

The project is created and opens in a window according to your choice .

Creating Kotlin-JavaScript Project

To create a Kotlin-JavaScript project, follow these steps

1. On the main menu, choose File | New | Project.
2. In the New Project dialog box, select the project type Kotlin-JavaScript.
3. Click Next.
4. On the next page of the wizard, specify the following:
   - Project name and location
   - Project SDK. Use one of the options from the drop-down list, or, if the required SDK is missing, click New.
   - Kotlin runtime library. Select the desired library from the drop-down list, or, if missing, click Create.

   If necessary, click ▸ (More settings)and specify the settings as required.
5. Click Finish

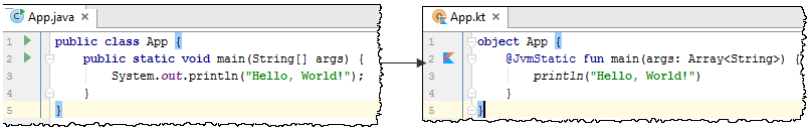
The project is created and opens in a window according to your choice .

Converting a Java File to Kotlin File

IntelliJ IDEA enables you to convert a Java file to Kotlin.

To convert Java file to Kotlin

1. On the main menu, point to Code menu.
2. Choose Convert Java File to Kotlin File.



Note that the resulting Kotlin file appears in place of the original Java file.

Mixing Java and Kotlin in One Project

On this page:

– Creating Java files in Kotlin projects

– Creating Kotlin files in Java projects

Creating Java files in Kotlin projects

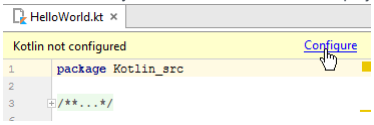
To create a Java file in a Kotlin project, follow these steps

1. Create a Kotlin project as described in the sections Creating Kotlin-JVM Project or Creating Kotlin-JavaScript Project .
2. In the Project tool window , select the target package or directory, where you want the Java class to be created.
3. Press `Alt+Insert` , choose Java Class from the pop-up menu, select its kind, and specify its name. See Creating Java Classes, Interfaces, Enumerations and Annotations .

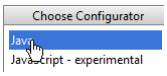
Creating Kotlin files in Java projects

To create a Kotlin file in a Java project, follow these steps

1. In a Java project, select the target location, press `Alt+Insert` and choose Kotlin file from the pop-up menu.
2. If this is first time you create a Kotlin file in a Java project, the banner shows:



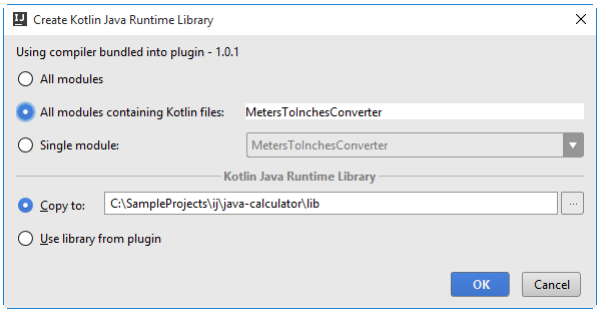
Click the link Configure, and select the desired configurator:



In the dialog box Create Kotlin Java Runtime Library that opens, do the following:

– Choose the modules to configure (if there are several modules in a project)

– Specify which Kotlin run-time library should be used.



Note that in the future the configuration action is not required.

Markup Languages and Style Sheets

In this section:

- Markup Languages and Style Sheets
  - Markup and template languages support
  - Parsing Web contents
- Changing Color Values in Style Sheets
- CSS-Specific Refactorings
- Editing HTML Files
- Emmet Support
- Generating DTD
- Generating Instance Document from XML Schema
- Generating XML Schema from Instance Document
- Live Editing of HTML, CSS, and JavaScript
- Minifying CSS
- Referencing XML Schemas and DTDs
- Transpiling Sass, Less, and SCSS to CSS
- Using Handlebars and Mustache Templates
- Using JetBrains Chrome Extension
- Validating Web Content Files
- Viewing Actual HTML DOM
- Viewing Images
- Viewing Styles Applied to a Tag
- Working with Sass and SCSS in Compass Projects
- XML-Java Binding

Note In this part you will find information that is specific for the web content files only!

Markup and template languages support

IntelliJ IDEA supports editing of files in the following markup and template languages:

- XML
- HTML/XHTML
- JSP/JSPX
- CSS
- Sass, SCSS
- Slim
- Less
- YAML
- Stylus
- Compass
- Handlebars expressions and Mustache templates

The markup languages and style sheets are integrated into IntelliJ IDEA and can use the most powerful editing features:

- Validation and syntax highlighting.
- Code completion (`Ctrl+Space`).
- Indentation (`Ctrl+Alt+I` , `Ctrl+Alt+L`).
- Formatting (`Ctrl+Alt+L`) according to the code style .
- Intention actions (`Alt+Enter`).
- Viewing code structure (`Alt+7`).
- Navigation in the source code (`Ctrl+B`).
- Integrated documentation (`Ctrl+Q`).
- Search for usages (`Alt+F7`).
- Commenting and uncommenting lines (`Ctrl+Slash` , `Ctrl+Shift+Slash`).
- Unwrapping and removing tags (`Ctrl+Shift+Delete`).

All these features work if IntelliJ IDEA successfully locates the DTD or schema file. In this case, all the files are validated against the DTD or schema, and the editing conveniences become available. Without a DTD or schema, only the well-formedness check is possible.

These features for web contents work same way as for the other source files. Refer to the respective topics of the Advanced Editing Features part for the detailed descriptions of procedures, and to Keyboard shortcuts .

Parsing Web contents

IntelliJ IDEA parses Web contents files according to the following specifications:

- HTML: specification is configurable in the Default HTML language level in the Schemas and DTDs page of the Settings/Preferences dialog. By default, specification HTML 5.0 from W3C is assumed.
- CSS: specification CSS 3.0. The most common selectors are supported: universal selector `*` , type selectors `.a` , descendant selectors `.a.b` , child selectors `.a .b` , ID selectors `#b` , pseudo-classes and class selectors `DIV.warning` .
- IntelliJ IDEA uses Xerces 2.11, an XML parser developed by Apache Software Foundation Group.

Changing Color Values in Style Sheets

On this page:

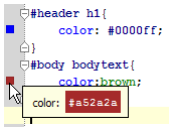
– Introduction
– Choosing color value
– Changing color value

Introduction

In addition to the editing techniques that are common for all file types, IntelliJ IDEA provides specific techniques for CSS, Sass, SCSS and Less files.

In particular, it is possible to easily change color values without the necessity to memorize and type color codes.

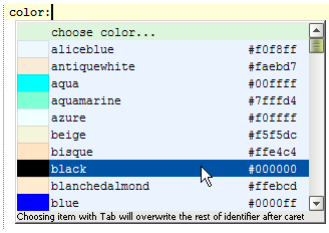
The `color` properties are marked with the icons of the corresponding color in the left gutter area of the editor. Use these icons to view colors and change color values.
When you point with your mouse cursor to the color icon, the pop-up window appears, showing the color and its code.



Choosing color value

To choose color value in a style sheet

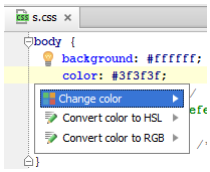
1. Open the desired style sheet for editing.
2. Type `color:` , and then press `Ctrl+Space` .
3. Select the desired color value from the suggestion list, or choose color... to pick a custom one:



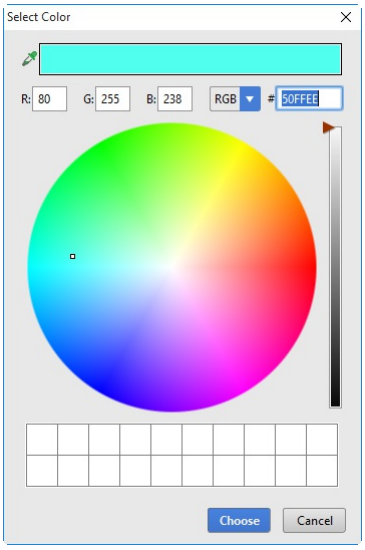
Changing color value

To change color value in a style sheet

1. Open the desired style sheet for editing, and locate color property to be changed.
2. Do one of the following:
   – If the color icon is shown , just double-click is in the left gutter of the editor.
   – If the color icon is not shown , then press `Alt+Enter` , or click to reveal the list of intention actions, and choose Change color intention action:



3. In the Choose color dialog box, pick the desired new color, and click Choose:



CSS-Specific Refactorings

In addition to the full range of common refactorings , IntelliJ IDEA provides the following CSS-specific refactorings:

- Extract Variable for Sass

Extract Variable for Sass

You can replace a Sass expression with a local or global variable.

To perform this refactoring, you can use:

- In-place refactoring . In this case you specify the new name right in the editor.
- Refactoring dialog , where you specify all the required information. To make such a dialog accessible, you have to clear the check box Enable in-place mode in the editor settings.

Example

Before After

```
$blue: #3bbfce
                    $margin: 16px

                    .border
                    padding: $margin / 2
                    margin: $margin / 2
                    border-color: $blue
```

```
$blue: #3bbfce
                    $margin: 16px;
                    $var: $margin / 2

                    .border
                    padding: $var
                    margin: $var
                    border-color: $blue
```

To extract a variable using in-place refactoring

1. In the editor, select the expression to be replaced with a variable. You can do that yourself or use the smart expression selection feature to let IntelliJ IDEA help you. So, do one of the following:
   - Highlight the expression. Then choose Refactor | Extract | Variable on the main menu or on the context menu. Alternatively, press `Ctrl+Alt+V`.
   - Place the cursor before or within the expression. Choose Refactor | Extract Variable on the main menu or on the context menu. or press `Ctrl+Alt+V`. In the Expressions pop-up menu, select the expression. To do that, click the required expression. Alternatively, use the Up and Down arrow keys to navigate to the expression of interest, and then press `Enter` to select it.

   Note The Expressions pop-up menu contains all the expressions appropriate for the current cursor position in the editor.

   When you navigate through the suggested expressions in the pop-up, the code highlighting in the editor changes accordingly.

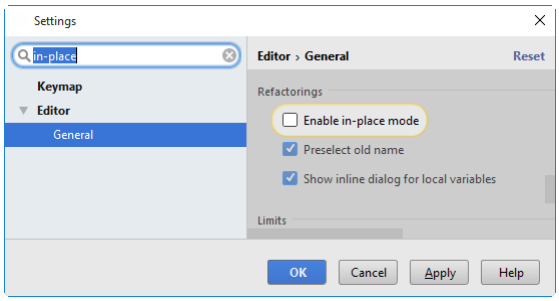
2. If more than one occurrence of the selected expression is found, select Replace this occurrence only or Replace all occurrences in the Multiple occurrences found pop-up menu. To select the required option, just click it. Alternatively, use the `Up` and `Down` arrow keys to navigate to the option of interest, and press `Enter` to select it.

3. Select the place in the source code, where the new variable will be declared. The declaration can be global (a variable is available throughout the whole file), or local (a variable is declared immediately before use, and is available in the current block only).

4. Specify the name of the variable. Do one of the following:
   - Select one of the suggested names from the pop-up list. To do that, double-click the suitable name. Alternatively, use the `Up` and `Down` arrow keys to navigate to the name of interest, and `Enter` to select it. When finished, press `Escape`.
   - Edit the name by typing. The name is shown in the box with red borders and changes as you type. When finished, press `Escape`.

To extract a variable using the dialog box

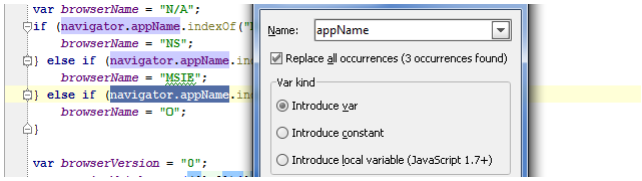
If the Enable in place refactorings check box is cleared in the Editor settings, the Extract Variable refactoring is performed by means of the dialog box.



1. Select the desired expression, and invoke Extract Variable refactoring as described above .

2. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.

3. In the Extract Variable dialog for Sass :
   - Specify the variable name. You can select one of the suggested names from the list or type the name in the Name field.
   - Specify the place for declaration. Select the desired place (global or local) from the drop-down list.
   - If more than one occurrence of the selected expression is found, you can select to replace all the found occurrences by selecting the corresponding check box. If you want to replace only the current occurrence, clear the Replace all occurrences check box.
   - Click OK.

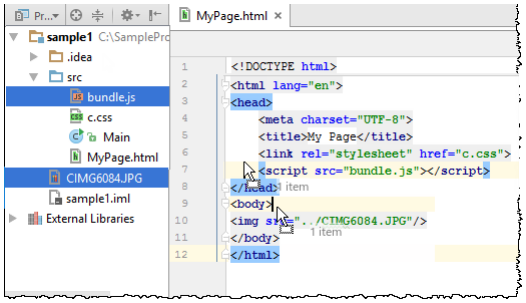

var browserName = "N/A";
if (navigator.appName.indexOf("
browserName = "NS";
} else if (navigator.appName.in
browserName = "MSIE";
} else if (navigator.appName.in
browserName = "O";
}
var browserVersion = "0";
Name:
appName
Replace all occurrences (3 occurrences found)
Var kind
Introduce var
Introduce constant
Introduce local variable (JavaScript 1.7+)

Editing HTML Files

IntelliJ IDEA enables you to create HTML files , and edit them.

When a file is opened for editing, you can drag a JavaScript or CSS file into the opened HTML file. IntelliJ IDEA will automatically create a script or a link tag inside the `<head>` tag.

Same way, you can drag an image to add an `img` tag with the `width` and `height` attributes anywhere inside the `<body>` tag.



Emmet Support

With IntelliJ IDEA, you can edit HTML and CSS code faster by applying Emmet features. IntelliJ IDEA provides two types of Emmet support.

- Native
- Support of additional templates

In this section:

- Enabling Emmet Support
- Enabling Support of Additional Live Templates
- Configuring Abbreviation Expansion Key
- Expanding Emmet Templates with User Defined Templates
- Surrounding a Code Block with an Emmet Template
- Navigating Between Edit Points

Enabling Emmet Support

On this page:

- Basics
- Enabling and Configuring Native Emmet Support in the HTML or XML Context
- Enabling Native Emmet Support in the JavaScript Context
- Enabling and Configuring Native Emmet Support in the CSS Context

Basics

Native Emmet support allows you to generate XML/HTML, JavaScript (JSX Harmony) and CSS structures based on abbreviations . IntelliJ IDEA supports basic Emmet and Emmet version 1.1 features, such as:

- New syntax for writing RGBA colors.
- Implied attributes.
- Default attributes.
- Boolean attributes.
- The Update Tag action.

Emmet is supported in HTML/XML, JavaScript (JSX Harmony) and in the CSS contexts. This support is configured separately on the HTML , JSX and CSS pages respectively.

Enabling and Configuring Native Emmet Support in the HTML or XML Context

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click XML under Emmet. The Emmet page opens.
2. To enable the Emmet support in the HTML or XML context, select the Enable XML/HTML Emmet check box.
   When this check boxes is cleared, all the other controls on this page become disabled.
3. To have IntelliJ IDEA show a pop-up window with a preview of the entered abbreviation before actually expanding it , select the Enable abbreviation preview check box.
4. Specify how Emmet in IntelliJ IDEA will treat URL addresses by selecting or clearing the Enable automatic URL recognition while wrapping text with <a> tag check box.
   - If this check box is cleared and you attempt to wrap an URL address with the <a> tag, IntelliJ IDEA simply encloses the URL address in `<a href=""></a>` and positions the cursor inside the double quotes in the `href` attribute. For example, wrapping `http://www.jetbrains.com` will result in `<a href="|">http://www.jetbrains.com</a>` :

     ```
     <a href="|">http://www.jetbrains.com</a>
     ```

   - If this check box is selected and you attempt to wrap an URL address with the <a> tag, IntelliJ IDEA inserts the URL address inside the double quotes as the value of the `href` attribute and encloses the URL in `<a href="<wrapped URL>"></a>` . For example, wrapping `http://www.jetbrains.com` will result in `<a href="http://www.jetbrains.com">http://www.jetbrains.com</a>` . Moreover, IntelliJ IDEA highlights the wrapped URL green as a recognized URL:

     ```
     <a href="http://www.jetbrains.com">http://www.jetbrains.com</a>
     ```

Enabling Native Emmet Support in the JavaScript Context

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.
2. Under the Languages and Frameworks node, click JavaScript , and select the language level JSX Harmony.
3. Expand the Editor node, and then click JSX under Emmet. The JSX page opens.
4. To enable the Emmet support in the JavaScript context, select the Enable JSX Emmet check box.

Enabling and Configuring Native Emmet Support in the CSS Context

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click CSS under Emmet. The CSS page opens.
2. To enable the Emmet support in the CSS context, select the Enable CSS Emmet check box.
   When this check boxes is cleared, all the other controls on this page become disabled.
3. Configure the way unknown abbreviations are treated by selecting or clearing the Enable fuzzy search among CSS abbreviations check box: When this check box is selected, every unknown abbreviation will be scored against available template names. The match with the best score will be used to resolve the template. For example, with this option enabled, the following abbreviations can be equal to:
   - `ov:h`
   - `ov-h`
   - `o-h`
   - `oh`
4. Configure the way unrecognized properties are treated by selecting or clearing the Enable expansion of unknown properties ('unknown' to 'unknown:;') check box:
   - When this check box is selected, any entered word will be expanded into the same word followed with a colon and a semicolon;
   - When this check box is cleared, only known properties (for example, `color`) will be expanded this way (`color:;`)
5. Configure inserting browser-specific prefixes using the Auto insert CSS vendor prefixes check box: If this check box is selected, the CSS properties listed in the table below are expanded into constructs that contain pre-pending vendor prefixes. Learn more at Vendor prefixes .
   If this check box is cleared, the entire table of properties is disabled.
6. Configure the use of properties in different browsers using the Properties and vendor prefixes table. The table contains a list of CSS properties and vendor prefixes that correspond to various browsers.
   - To enable or disable a property in a browser, select or clear the check box under the browser column.
   - To add a new property to the list, click the Add button + or press `Alt+Insert` . Then type the name of the property in the dialog box that opens and enable it in the relevant browsers.
   - To delete one or more properties from the lsit, select them and press Remove – or press `Alt+Delete` .

Enabling Support of Additional Live Templates

Support of additional templates includes more than 200 different HTML, CSS, and XSL live templates. All of them are listed under the Emmet nodes on the Live Templates page of the Settings dialog box.

To enable support of additional HTML, CSS, and XSL live templates

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Live Templates under Editor.

2. On the Live Templates page, that opens, select the check boxes next to the relevant template groups.

Configuring Abbreviation Expansion Key

Shortcut key for expanding Emmet selectors and live templates is configurable. You can re-define this default setting for each specific live template.

Configuring a shortcut to expand abbreviations

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Emmet under Editor.

2. On the Emmet page that opens, choose the desired option from the Expand abbreviation with drop-down list.

Configuring a shortcut to expand a live template with

1. To re-define the expansion key for a live template, open the Live Templates page, expand one of the Zen Coding nodes, and select the desired template. The focus moves to the Template Text area.

2. From the Expand with drop-down list, select the key to expand the template with.
   This setting does not override the default setting specified for native Emmet support; you will just get the ability to expand the template using either of the specified keys.

Expanding Emmet Templates with User Defined Templates

You can expand Emmet templates with your own live templates.

Suppose you have a template entry with the following template text:

```
<entry type="$TYPES$">$END$ <entry>
```

To generate a list of entries, you just need to type `"entry-list<entry[number=$]*5"` and press TAB. By default, the `number` attribute will be generated before `type`.
To customize the position where it is generated, you need to add the `ATTRS` variable to your template, for example:

```
<entry type="$TYPES$"
                    $ATTRS$>$END$ <entry>
```

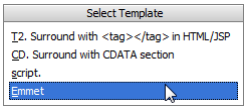
The `ATTRS` variable must have an empty string as the default value and should be skipped.

To expand an Emmet template with a user-defined template

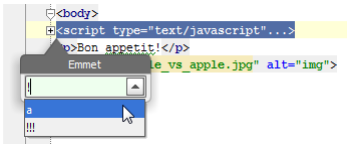
1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Live Templates under Editor.
2. On the Live Templates page that opens, expand the required Zen Coding template group, for example, `Zen HTML`, Zen CSS, or Zen XSL.
3. Select the template to expand. The focus moves to the Template Text area where the fields show the settings of the selected template.
4. In the Template Text field, add the required text and variables to the template body.
5. Click the Edit Variables button. In the Edit Template Variables dialog box that opens, specify the default variable values in the Default value field and select the Skip if defined check box, where necessary.

To surround a block of code with an Emmet template, follow these steps

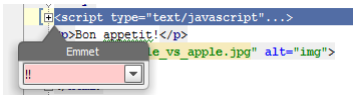
1. Open the desired file for editing and select a block of code to be surrounded.
2. Press `Ctrl+Alt+J`, or choose Code | Surround with Live Template on the main menu .
3. In the Select Template pop-up menu, choose Emmet:



4. Type the desired Emmet abbreviation, and press `Enter`.
   Note the drop-down list to the right. Clicking the down arrow reveals a history list of the recently applied Emmet live templates:



   Also mind the color indication. If you type a valid Emmet abbreviation, the background is green. However, when a non-existent abbreviation is entered, the background becomes red:



Navigating Between Edit Points

In the HTML and XML context, you can navigate between edit points, that is, between important points of code where editing is possible.

- To move the cursor to the previous edit point, choose Navigate | Previous Emmet Edit Point, or press `Shift+Alt+Open Bracket`.
- To move the cursor to the next edit point, choose Navigate | Next Emmet Edit Point, or press `Shift+Alt+Close Bracket`.

Generating DTD

A Data Type Definition (DTD) is required for running structure validation checks on a Web content file. IntelliJ IDEA can scan any XML file for the existing elements and attributes and generate a DTD for it.

To generate a DTD for an XML file

1. Open the desired file in the active editor tab.
2. On the main menu, choose Tools | XML Actions | Generate DTD From XML File. The resulting DTD is added as a section above the first line of the file.

To generate an XML instance document from an XML Schema

1. With the desired Schema (`.xsd`) file opened in the active editor tab, choose Tools | XML Actions | Generate XML Document from XSD Schema on the main menu.

2. In the Generate Instance Document from Schema dialog box that opens configure the XML instance document generation procedure:
   - In the Schema Path text box, specify the location of the Schema to base the XML document generation on. By default, the field shows the full path to the current file. Accept this suggestion or click the Browse button and select the desired file in the dialog that opens .
   - In the Instance Document Name text box, specify the name of the output file to place the generated XML in.

     Warning! IntelliJ IDEA suggests the name of the source XML document with the `.xml` extension. If you type another name, make sure the extension is correct.
   - Specify the location of the generated document. By default, it will be placed in the same directory as the source Schema file. To specify another location, click the Browse button and select the desired path in the dialog that opens .
   - From the Element Name drop-down list, select the local name of the global element to be used as the root of the generated XML document.
   - Specify whether to take restriction and uniqueness particles into consideration by selecting the corresponding check boxes.

Generating XML Schema From Instance Document

An XSD (XML Schema Definition) Schema is required for running structure validation checks on a Web content file. IntelliJ IDEA can scan any XML file for the existing elements and attributes and generate a Schema for it.

To have a Schema generated based on an XML instance document

1. With the desired XML document opened in the active editor tab, choose Tools | XML Actions | Generate XSD Schema from XML File on the main menu.

2. In the Generate Schema From Instance Document dialog box that opens configure the Schema generation procedure:
   - In the Instance Document Path text box, specify the location of the file to be used as the basis for Schema generation. By default, the field shows the full path to the current file. Accept this suggestion or click the Browse button and select the desired file in the dialog that opens .
   - In the Result Schema File Name text box, specify the name of the output file to place the generated Schema in.

     Warning! IntelliJ IDEA suggests the name of the source XML document with the `.xsd` extension. If you type another name, make sure the extension is correct.
   - Specify the location of the generated Schema. By default, the generated Schema file will be placed in the same directory as the source XML instance document. To specify another location, click the Browse button and select the desired path in the dialog that opens .
   - From the Design Type drop-down list, select the way to declare elements and complex types.
   - From the Detect Simple Content Type drop-down list, select the type to use for leaf text.
   - In the Detect Enumerations Limit text box, type the number of occurrences to cause the Schema enumeration appearance.

     Tip To suppress Schema enumeration, specify 0.

Live Editing of HTML, CSS, and JavaScript

On this page:

- Introduction
- Before you start
- Configuring the Live Edit update policy in the debugging mode
- Live Editing during a debugging session

Introduction

During a JavaScript debugging session, you can monitor how the changes you make in the HTML file are rendered in the actual browser window without refreshing the page manually and even without leaving IntelliJ IDEA. All the changes you make are immediately reflected in the browser and the affected area is highlighted. Besides pure HTML files, this also works for other file types that contain or generate HTML, CSS, or JavaScript. You just need to specify the URL of the page in the `JavaScript Debug` run configuration. The live contents of the page being edited are shown in the `Elements` tab of the Debug tool window .

Currently the Live Edit functionality is supported only for Google Chrome and only during a debugging session.

Before you start

1. Install and enable the LiveEdit plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Make sure the JetBrains Chrome Extension is installed in your Chrome browser.
   The extension is installed when you first initiate a debugging session and follow the dedicated link provided by IntelliJ IDEA. For details, see Using JetBrains Chrome Extension .

Configuring the Live Edit update policy in the debugging mode

In the debugging mode, IntelliJ IDEA provides the Live Edit functionality which supports both automatic and manual upload of updated files on the server side. For the client side, Live Edit does not provide any way to apply the changes.

In Meteor projects, the client-side code can be updated through the native Meteor hot code pushes functionality . See Using Meteor for details.

To configure the Live Edit update policy in the debugging mode

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Debugger node under Build, Execution, Deployment, and then click Live Edit. The Live Edit page opens.

2. In the Update area, configure the way changes made to the code during a debugging session are applied. Note that an update will now be performed only if none of the modified files have any syntax errors.
   - Auto in (ms): Choose this option to have the changes applied automatically at certain time interval and specify the delay in ms in the text box. This policy is not available for the client-side code of Meteor applications.
   - Manual: Choose this option to apply the changes manually by clicking the Update <run configuration name> JavaScript button or the Update <run configuration name> button on the toolbar of the Debug tool window.
     It is recommended that you choose this option for debugging Meteor applications because applying changes to the client-side code is not supported.
   - Restart if hotswap fails:
     With changes in HTML, CSS, and JavaScript on the client side, the contents of a Web page in the browser are updated without reloading. For NodeJS or Meteor applications, IntelliJ IDEA first tries to update the application incorporating the changes without restarting the NodeJS server.
     - Select this check box to have IntelliJ IDEA try restarting the server if the changes cannot be applied automatically. After the restart, IntelliJ IDEA brings you to the execution point where the problem took place.
       If even with this option chosen automatic upload still fails, tou will have to restart the server manually by clicking the Rerun <run configuration name> button .
     - When the check box is cleared, IntelliJ IDEA just displays a pop-up window informing you about the failure and suggesting you to restart the server manually.

Live Editing during a debugging session

Live Editing is enabled automatically, no steps are required from your side to activate it.

To perform Live Editing during a debugging session

1. When editing the code and monitoring how the changes are rendered in the browser, it is helpful to have the areas affected by the changes highlighted. To enable highlighting affected areas, open the IntelliJ IDEA settings dialog box, click Live Edit, and then select the Highlight current element in browser check box on the Live Edit page that opens.

2. To initiate a debugging session, create a run configuration of the type JavaScript Debug and click the Debug button on the toolbar. IntelliJ IDEA establishes connection with the JetBrains Chrome extension and starts a debugging session.

3. Edit the HTML file that implements the page opened in the browser and view the changes immediately rendered.

IntelliJ IDEA supports an operation that is somehow opposite to live editing. During a debugging session, the Elements tab of the Debug tool window shows the HTML source code that implements the actual browser page and its HTML DOM structure . Moreover, any changes made to the page through the browser are immediately reflected in the Elements tab. For more details, see Viewing Actual HTML DOM .

Minifying CSS

On this page:

- Introduction
- Installing and configuring the YUI Compressor
- Creating a File Watcher
- Minifying the code

Introduction

The term minification or compression in the context of CSS means removing all unnecessary characters, such as spaces, new lines, comments without affecting the functionality of the source code.

These characters facilitate working with the code at the development and debugging stage by improving the code readability. However at the production stage these characters become extraneous: being insignificant for code execution, they increase the size of code to be transferred. Therefore it is considered good practice to remove them before deployment.

IntelliJ IDEA supports integration with the YUI Compressor CSS minification tool.

In IntelliJ IDEA, minifier configurations are called File Watchers. For each supported minifier, IntelliJ IDEA provides a predefined File Watcher template. Predefined File Watcher templates are available at the IntelliJ IDEA level. To run a minifier against your project files, you need to create a project-specific File Watcher based on the relevant template, at least, specify the path to the minifier to use on your machine.

Installing and configuring the YUI Compressor

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The CSS minifier is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the CSS minifier.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the CSS minifier and npm from any folder.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Download and install the JavaScript minification tool. The easiest way is to use the Node Package Manager (npm), which is a part of Node.js .
   1. Switch to the directory where the Node Package Manager (npm) is stored or define a `path` variable for it so it is available from any folder.
   2. Type the following command at the command line prompt:

```
npm install yuicompressor
```

   If you use the Node Package Manager (npm), the minifier is installed under Node.js so Node.js, which is required for starting the tool, will be specified in the path to it.

Creating a File Watcher

IntelliJ IDEA provides a common procedure and user interface for creating File Watchers of all types. The only difference is in the predefined templates you choose in each case.

1. To start creating a File Watcher, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X on the main menu, and then click File Watchers under the Tools node. The File Watchers page that opens, shows the list of File Watchers that are already configured in the project.
2. Click the Add button + or press `Alt+Insert` and choose the YUI Compressor CSS predefined template from the pop-up list.
3. In the Program text box, specify the path to the `yuicompressor-<version>.jar` file. If you installed the tool through the Node Package Manager, IntelliJ IDEA locates the required file itself and fills in the field automatically. Otherwise, type the path manually or click the Browse button ... and choose the file location in the dialog box that opens.
4. Proceed as described on page Using File Watchers .

Minifying the code

When a Minification File Watcher is enabled (see Enabling and disabling File Watchers), minification starts automatically as soon as a file to which transpilation is applicable is changed or saved, see Configuring the behaviour of the File Watcher .

IntelliJ IDEA creates a separate file with the generated output. The file has the name of the source CSS file and the extension `min.css` . The location of the generated file is defined in the Output paths to refresh text box of the New Watcher dialog . However, in the Project Tree, it is shown under the source CSS file which is now displayed as a node.

Referencing XML Schemas and DTDs

Your XML file may reference an external XML schema (XSD) or DTD file, e.g.

```
<root   xmlns="http://www.example.org"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.example.org
                    http://www.example.org/xsds/example.xsd">
```

or

```
<!DOCTYPE root SYSTEM "http://www.example.org/dtds/example.dtd">
```

If the referenced URL or the namespace URI is "unfamiliar", it's marked as an error.

For situations like this, IntelliJ IDEA provides the following intention actions :

- Fetch External Resource. IntelliJ IDEA downloads the referenced file and associates it with the URL (or the namespace URI). The error highlighting disappears. The XML file is validated according to the downloaded schema or DTD. (The associations of the URLs and the namespace URIs with the schema and DTD files are shown on the Schemas and DTDs page in the Settings dialog .)
- Manually Setup External Resource. Use this option when you already have an appropriate schema or DTD file available locally. The Map External Resource dialog will open and you'll be able to select the file for the specified URL or namespace URI. The result of the operation is the same as in the case of fetching the resource.
- Ignore External Resource. The URL or the namespace URI is added to the Ignored Schemas and DTDs list. (This list is shown on the Schemas and DTDs page in the Settings dialog .) The error highlighting disappears. IntelliJ IDEA won't validate the XML file, however, it will check if the XML file is well-formed.

There is one more intention action that you may find useful: Add Xsi Schema Location for External Resource. This intention action lets you complete your root XML elements. If the namespace is already specified, IntelliJ IDEA can add a couple of missing attributes.

For example, if you have a fragment like this:

```
<root   xmlns="http://www.example.org">
```

and perform the intention action on the value of the `xmlns` attribute, the result will be:

```
<root   xmlns="http://www.example.org"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.example.org >
```

At this step, you'll be able to add the schema URL, and then map the URL (or the namespace URI) onto an appropriate schema file, or add the URL (or the URI) to the Ignored Schemas and DTDs list.

Transpiling Sass, Less, and SCSS to CSS

On this page:

- Introduction
- Installing the Sass/SCSS transpiler
- Installing the Less transpiler
- Installing the Less transpiler globally
- Installing the Less transpiler in a project
- Creating a File Watcher
- Transpiling the code

Introduction

Sass, Less, and SCSS code is not processed by browsers that work with CSS code. Therefore to be executed, Sass, Less, or SCSS code has to be translated into CSS. This operation is referred to as transpilation and the tools that perform it are called transpilers.

IntelliJ IDEA supports integration with transpiler tools that translate Sass, Less, and SCSS code into CSS.

In IntelliJ IDEA, transpiler configurations are called File Watchers. For each supported transpiler, IntelliJ IDEA provides a predefined File Watcher template. Predefined File Watcher templates are available at the IntelliJ IDEA level. To run a transpiler against your project files, you need to create a project-specific File Watcher based on the relevant template, at least, specify the path to the transpiler to use on your machine.

Warning! Make sure the NodeJS plugin is enabled. You need to download it form the JetBrains plugin repository and activate it on the Plugins page of the Settings/Preferences dialog box as described in Enabling and Disabling Plugins.

Installing the Sass/SCSS transpiler

Sass and SCSS transpilers are managed through the Ruby Gem manager.

1. Download and install Ruby .
2. Specify a `path` variable for the folder where the Ruby executable file and the `gem.bat` file are stored. This lets you launch Ruby and Gem Manager from any folder and ensures that Ruby is successfully launched during transpilation.
3. Type one of the following commands at the command line prompt:

```
gem install sass
```

or

```
gem install scss
```

The tool is installed to the folder where Ruby executable file and the `gem.bat` file are stored.

Installing the Less transpiler

The easiest way to install the Less transpiler is to use the Node Package Manager (npm), which is a part of Node.js . See Installing and Removing External Software Using Node Package Manager for details.

Depending on the desired location of the Less transpiler executable file, choose one of the following methods:

- Install the transpiler globally at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project.
- Install the transpiler in a specific project and thus restrict its use to this project.
- Install the transpiler in a project as a development dependency .

In either installation mode, make sure that the parent folder of the Less transpiler is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

IntelliJ IDEA provides user interface both for global and project installation as well as supports installation through the command line.

Installing the Less transpiler globally

Global installation makes a transpiler available at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project. Moreover, during installation the parent folder of the transpiler is automatically added to the `PATH` variable, which enables you to launch the transpiler from any folder. To install the transpiler globally, do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
  3. Type the following command at the command line prompt:

```
npm install -g less
```

  The `-g` key makes the transpiler run in the global mode. Because the installation is performed through NPM, the Less transpiler is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

  For more details on the NPM operation modes, see npm documentation . For more information about installing the Less transpiler, see https://npmjs.org/package/less .

- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the

global and at the project level. Click +.

3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing the Less transpiler in a project

Installing a transpiler in a specific project restricts its use to this project. To run project installation, do one of the following:

– Run the installation from the command line:

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install less
```

– Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in template-based projects of the type Node Boilerplate or Node.js Express, which already have the `node_modules` folder. The latter is important because NPM installs the Less transpiler in a `node_modules` folder. If your project already contains such folder, the Less transpiler is installed there.

Projects of other types or empty projects may not have a `node_modules` folder. In this case npm goes upwards in the folder tree and installs the Less transpiler in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be outside your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the Less transpiler is installed there.

In either case, make sure that the parent folder of the Less transpiler is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

Creating a File Watcher

IntelliJ IDEA provides a common procedure and user interface for creating File Watchers of all types. The only difference is in the predefined templates you choose in each case.

1. Install and enable the File Watchers repository plugin.
   The plugin is not bundled with IntelliJ IDEA, but it is available from the IntelliJ IDEA plugin repository plugin repository . See Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins for details.
2. To start creating a File Watcher, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X on the main menu, and then click File Watchers under the Tools node. The File Watchers page that opens, shows the list of File Watchers that are already configured in the project.
3. Click the Add button + or press `Alt+Insert` . Depending on the tool you are going to use, choose the appropriate predefined template from the pop-up list:
   – Less
   – Sass
   – SCSS
4. In the Program text box, specify the path to the compiler executable file or archive depending on the chosen predefined template.
   – `lessc.cmd` for Less If you installed the tool through the Node Package Manager, IntelliJ IDEA locates the required file itself at `<node.js_home>/node_modules/bin/lessc.cmd` and fills in the field automatically. Otherwise, type the path manually or click the Browse button ... and choose the file location in the dialog box that opens.
   – `sass.bat` for Sass
   – `scss.bat` for SCSS

   If you installed the Sass and SCSS tools through Ruby, IntelliJ IDEA locates the required files itself at `<ruby_home>/bin/sass.bat` `<ruby_home>/bin/scss.bat` respectively and fills in the field automatically. Otherwise, type the path manually or click the Browse button ... and choose the file location in the dialog box that opens.
5. Proceed as described on page Using File Watchers .

Transpiling the code

When you open a TypeScript file, IntelliJ IDEA checks whether an applicable file watcher is available in the current project. If such file watcher is configured but disabled, IntelliJ IDEA displays a pop-up window that informs you about the configured file watcher and suggests to enable it.

If an applicable file watcher is configured and enabled in the current project, IntelliJ IDEA starts it automatically upon the event specified in the New Watcher dialog .

– If the Immediate file synchronization check box is selected, the File Watcher is invoked as soon as any changes are made to the source code.
– If the Immediate file synchronization check box is cleared, the File Watcher is started upon save (File | Save All, `Ctrl+S`) or when you move focus from IntelliJ IDEA (upon frame deactivation).

IntelliJ IDEA creates a separate file with the generated output. The file has the name of the source Sass, Less, or SCSS file and the extension `css` . The location of the generated files is defined in the Output paths to refresh text box of the New Watcher dialog . However, in the Project Tree, they are shown under the source file which is now displayed as a node.

Using Handlebars and Mustache Templates

On this page:

– Basics
– Preparing to use Handlebars expressions and Mustache templates
– Configuring processing of Handlebars expressions and Mustache templates

Basics

With IntelliJ IDEA, you can use Handlebars expressions and Mustache templates in dedicated Handlebars and Mustache files that have the extension `.hbs` or `.mustache` respectively. IntelliJ IDEA distinguishes these file types and processes their contents according to default or custom settings specified on the Templates page of the Settings/Preferences dialog.

For more details about Handlebars expressions and Mustache templates see http://handlebarsjs.com/ .

The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Preparing to use Handlebars expressions and Mustache templates

To prepare using Handlebars expressions and Mustache templates

1. To enable the Handlebars/Mustache support in IntelliJ IDEA, install and activate the Handlebars/Mustache repository plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

2. Add the Handlebars and Mustache file types to IntelliJ IDEA:
   1. Open the Settings dialog by choosing File | Settings . Then select File Types. Find more on page Accessing Settings .
   2. On the File Types page that opens, click the Add button +.
   3. In the New File Type dialog box that opens, specify the name of the new type Handlebars/Mustache and optionally provide a description. When you click OK, IntelliJ IDEA returns to the File Types page.
   4. Select the newly registered file type and in the Registered Patterns area, specify the patterns that define the files of this type `*.hbs` and `*.mustache`. To add a pattern, click the Add button + and type the pattern in the Add Wildcard dialog box that opens.

   Find more on page Creating and Registering File Types .

3. Create a file of the type Handlebars or Mustache and place your code there. To create a file, select the parent folder and on the context menu of the selection choose New | Handlebars and specify the file name in the dialog box that opens. Find more on page Creating Template-based Files .

Configuring processing of Handlebars expressions and Mustache templates

To configure processing of Handlebars expressions and Mustache templates

The configuration settings you specify apply only to dedicated Handlebars and Mustache files that have the extension `.hbs` or `.mustache` respectively.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the JavaScript node under Languages&Frameworks, and then click Templates under JavaScript. The Templates page opens. Switch to the Handlebars/Mustache area.

2. Enable or disable tag completion.
   – To have IntelliJ IDEA automatically insert the second closing curly brace (`}`) of a Handlebars expression as soon as you type the first closing one, select the Automatically insert closing tag check box.
   IntelliJ IDEA also recognizes triple stashes (`{{{`) that prevent escaping values inside expressions. In this case, IntelliJ IDEA automatically inserts two closing curly braces as soon as you type the first closing one.
   – When this check box is cleared, you have to type the closing curly braces and triple stashes manually.

3. To have Handlebars expressions and Mustache templates automatically reformatted during code generation, refactoring, or reformatting (`Ctrl+Alt+L`), select the Enable formatting check box.
   If the check box is cleared, the original formatting of Handlebars expressions and Mustache templates is preserved.

4. From the Language for comments drop-down list, select the language to inherit the style for comments from. When you enter a line or block comment by pressing `Ctrl+Slash` or `Ctrl+Shift+Slash`, IntelliJ IDEA inserts the comment delimiters that are used in the chosen language, for example, `{{!----}}` for Handlebars, `/**/` for JavaScript, `<!---->` for HTML, etc.

Using JetBrains Chrome Extension

On this page:

- Introduction
- Installing the JetBrains Chrome extension
- Working with JetBrains Chrome extension
- Overriding the default CORS settings
- Changing port

Introduction

The JetBrains Chrome extension is mainly responsible for debugging JavaScript in Chrome and Dartium. During a debugging session, the extension also supports Live Editing of HTML, CSS, and other code that contains or produces JavaScript, and provides actual HTML DOM and dynamically updated source code of the actual page.

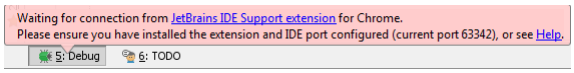
With the JetBrains Chrome extension, you can run several debugging sessions controlled by several IDEs and switch between them by changing the port which the extension listens to. And finally, the extension provides the possibility to override default CORS (Access-Control_Allow_Origin) settings and access local scripts on your machine that are by default protected.

Installing the JetBrains Chrome extension

You can install the extension during the first debugging session with Chrome or Dartium or at any time by visiting the Chrome Web Store.

To install the JetBrains Chrome extension

1. Do one of the following:
   - At any time open the Chrome Web Store at https://chrome.google.com/webstore/detail/jetbrains-ide-support/hmhgeddbohgjknpmjagkdomcpobmllji .
   - Start a JavaScript debugging session: create a run configuration with Chrome as the debugging browser, and click the Debug button on the toolbar. For details, see Debugging JavaScript . IntelliJ IDEA informs you that it is waiting for connection with the JetBrains IDE Support extension and shows the following message in the Debug tool window:

     Click the JetBrains IDE Support extension link which brings you to the Chrome Web Store.

   In either case the JetBrains IDE Support page opens where you can learn more about the extension.
2. Click the Add to Chrome button

.
3. In the Confirm New Extension dialog box that opens, click Add. The Add to Chrome button changes to Added to Chrome

.

When the extension is installed, the icon is displayed next to the Chrome address bar.

Working with JetBrains Chrome extension

Control over the JetBrains Chrome extension is provided through the chrome://extensions/ page.

To enable, disable, or remove the JetBrains Chrome extension

- To open the chrome://extensions/ page, click the Customize and control Chromium button next to the address bar, then choose Tools | Extensions on the context menu.
- To deactivate the extension, clear the Enabled check box. The check box name changes to Enable.
- To activate the extension, select the Enable check box.
- To uninstall the extension, click the Remove from Chromium button.

Overriding the default CORS settings

Suppose the page you are debugging requests a resource which is protected against access for security reasons through CORS settings. See more about CORS at https://developer.mozilla.org/en-US/docs/HTTP/Access_control_CORS .

You can enable access to the resources protected by default by changing the Chrome extension options.

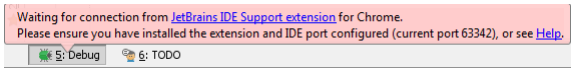
To override the default CORS settings

1. Right-click and choose Options on the context menu. A web page with Chrome extension options opens showing the parameters to connect to IntelliJ IDEA.
2. In the Force CORS text box, type the pattern that defines the URL addresses you want to make accessible, for example: `http://youtrack.jetbrains.com/rest/*` .

Changing port

During a debugging session, the Chrome extension listens to the port of the software product from which the extension was invoked. Each software product including IntelliJ IDEA has its own default port on which it starts by default.

If for some reason the default IntelliJ IDEA port is already busy, IntelliJ IDEA finds the closest available port and starts on it. This results in a conflict: IntelliJ IDEA is running on a "new" port while the Chrome extension still listens to the port of a previously started product and fails to establish connection with IntelliJ IDEA. The conflict reveals when you initiate a debugging session from IntelliJ IDEA: the extension fails to connect through the default port, IntelliJ IDEA waits for connection from the extension and displays the following message with the port number where it is actually running:



To fix the problem, specify the actual IntelliJ IDEA port in the Chrome extension options.

To change the port for connecting to IntelliJ IDEA

1. Right-click and choose Options on the context menu. A web page with Chrome extension options opens showing the parameters to connect to IntelliJ IDEA.

2. In the IDE Connection area, specify the actual IntelliJ IDEA port in the Port spin box.

Validating Web Content Files

On this page:

- Types of validity checks
- Choosing the default HTML language level
- Choosing the default schema to validate XML files
- Running full validation on an XML file

Types of validity checks

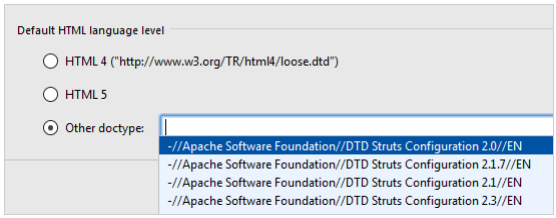
IntelliJ IDEA performs two different validity checks:

- On-the-fly validation is available for all Web content files and is performed as you edit the file. IntelliJ IDEA checks well-formedness, that is, detects various violations of syntax requirements, such as unclosed tags, wrong end-tag name, duplicate tags, unresolved links, etc. All encountered errors are highlighted in the editor.
  However, this form of code validation is rather *soft*, that is, not all requirements are taken into account.
- Full validation involves structure validation in addition to well-formedness check. Full validation is available for files that are associated with an XSD (XML Schema Definition) Schema or contain a Data Type Definition (DTD). IntelliJ IDEA checks whether the structure of your XML file complies with the structure defined in the corresponding DTD or Schema.
  The results of full validation are provided as a Message View.

Choosing the default HTML language level

Normally, an HTML or an XHTML file has the `<!DOCTYPE>` declaration which states the language level of the used in the source code from the file. This language level is used as a standard against which the contents of the file are validated. If an HTML or an XHTML file does not have a `<!DOCTYPE>` declaration, the contents of the file will be validated against the default standard (schema).

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Default XML Schemas under Schemas and DTDs.
   The Default XML Schemas page opens.
2. In the Default HTML Language Level area, choose the default schema to validate HTML and XHTML files without a `<!DOCTYPE>` declaration. The available options are:
   - HTML 4 or HTML 5: Choose one of these options to have files treated as HTML 4 or HTML 5 and validated against one of these standards.
   - Other doctype: Choose this option to have HTML files by default validated agains a custom DTD or schema and specify the URL of the DTD or schema to be used.
     Note that code completion is available in this field: press `Ctrl+Space` to see the list of suggested URLs.



   choose the XSD (XML Schema Definition) Schema to validate XML files. The available options are:
   - XML Schema 1.1 See W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures and W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes for details.
   - XML Schema 1.0 See XML Schema Part 1: Structures Second Edition and XML Schema Part 2: Datatypes Second Edition for details.

Choosing the default schema to validate XML files

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Default XML Schemas under Schemas and DTDs.
   The Default XML Schemas page opens.
2. In the Default XML Schema Version area, choose the XSD (XML Schema Definition) Schema to validate XML files. The available options are:
   - XML Schema 1.1 See W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures and W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes for details.
   - XML Schema 1.0 See XML Schema Part 1: Structures Second Edition and XML Schema Part 2: Datatypes Second Edition for details.

Running full validation on an XML file

1. Open the desired XML file in the editor, or just select it in the Project tool window.
2. On the context menu, choose Validate.

Viewing Actual HTML DOM

On this page:

- Introduction
- Before you start
- Viewing the HTML source and DOM structure

Introduction

Besides Live Editing, that is, monitoring how the changes you make in an HTML file are rendered in the actual browser window, during a debugging session you can perform a reverse operation using the Elements tab of the Debug tool window. The tab shows the HTML source code that implements the actual browser page and its HTML DOM structure . Moreover, any changes made to the page through the browser are immediately reflected in the Elements tab.

Before you start

1. Make sure you are using the Chrome browser for debugging.
2. Make sure the JetBrains Chrome Extension is installed in your Chrome browser.
   The extension is installed when you first initiate a debugging session and follow the dedicated link provided by IntelliJ IDEA. For details, see Using JetBrains Chrome Extension .
3. Make sure the LiveEdit repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Viewing the HTML source and DOM structure

To view the HTML source and DOM structure of the actual page

1. To initiate a debugging session, create a run configuration of the type JavaScript Debug and click the Debug button on the toolbar. IntelliJ IDEA establishes connection with the JetBrains Chrome extension and starts a debugging session.

2. Switch to the Debug tool window and open the Elements tab. The tab consists of three panes, all the panes are read-only. The Text pane shows the HTML source code of the page that is currently opened in the browser. As soon as any change is made to the page in the browser (e.g. clicking an icon), the code in the pane is updated accordingly.
   The Structure pane shows the DOM structure of the HTML code in the Text pane.

   The Scripts pane shows a tree of executed scripts.

   The Structure and Text panes are mutually synchronized. When you click a node in the DOM structure, IntelliJ IDEA scrolls through the contents of the Text pane.

   The panes are also synchronized with the browser. IntelliJ IDEA highlights the element in the browser as soon as you click the corresponding node in the DOM structure or in the Text pane.

Viewing Images

IntelliJ IDEA suggests several ways to view images, embedded in the HTML files. You can use navigation to source , open an image in an external graphical editor , or preview images on-the-fly . The viewing modes are configurable in the Images dialog .

To view an image in IntelliJ IDEA

1. In the Project tool window, select the desired image file.
2. On the context menu of the file, choose Jump to Source, or press `F4`.

Tip Configure the path to the desired external editor in the External Editor section of the Images dialog .

To view an image in an external editor

1. In the Project tool window, select an image file.
2. On the context menu of the file, choose Open in external editor, or press `Ctrl+Alt+F4`.

To preview an image, place the caret at a reference to an image in the editor, and do one of the following

- On the context menu, choose Jump to Source.
- Invoke the Navigate to declaration feature by pressing `Ctrl+B`.

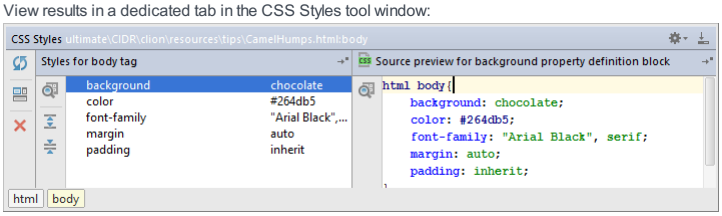
Viewing Styles Applied to a Tag

For HTML, XHTML, JSP, JSPX files, IntelliJ IDEA suggests a way to explore all styles applied to an arbitrary tag.

The results for each tag are displayed in the dedicated tabs of the CSS Styles tool window. Using this tool window, you can view the list of styles applied to a tag, and the definition of these styles. Besides that, you can navigate from style to the corresponding tag in the source code.

To show styles that are used for a tag

1. Open the desired file in the editor, and right-click the tag you want to explore for applied styles.
2. On the context menu, choose Show Applied Styles for Tag.
3. View results in a dedicated tab in the CSS Styles tool window:



Working with Sass and SCSS in Compass Projects

IntelliJ IDEA supports development in projects structured in compliance with the Compass framework . This framework uses Sass and SCSS extensions of CSS.

On this page:

- Preparing for Compass Development
- Setting up a Compass Project
- Integrating Compass with IntelliJ IDEA
- Creating a Compass Sass or a Compass SCSS Transpiler
- Running a Compass Sass or Compass SCSS Transpiler

Preparing for Compass Development

The Compass framework is installed through the Ruby Gem manager, therefore you need to install Ruby first.

1. Download and install Ruby .
2. Specify a `path` variable for the folder where the Ruby executable file and the `gem.bat` file are stored. This lets you launch Ruby and Gem Manager from any folder and ensures that Ruby is successfully launched during transpilation.
3. Install and enable the Sass Support plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
4. Install Compass.
   1. The installation is performed in the command line mode. To start the built-in Terminal, hover your mouse pointer over in the lower left corner of the IDE, then choose Terminal from the menu (see Working with Embedded Local Terminal for details).
   2. Type the following command at the command line prompt:

```
gem install compass
```

   The tool is installed to the folder where Ruby executable file and the `gem.bat` file are stored.

Setting up a Compass Project

You can have a project set up according to the Compass requirements in two ways: create a new Compass project or create an empty project and introduce a Compass-specific structure in it. In either case, a project is set up through command line commands. Of course, you can set up a Compass project externally and then open it in IntelliJ IDEA.

During project set-up, a `conf.rb` configuration file is generated. You will need to specify the location of this file when integrating Compass with IntelliJ IDEA.

- To set up the Compass-specific structure in an existing project:
  1. Open the desired project in IntelliJ IDEA.
  2. Open the built-in Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu.
  3. At the command line prompt, type:

```
compass init
```

- To create a Compass project from scratch:
  1. Open the desired project in IntelliJ IDEA.
  2. Open the built-in Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu.
  3. Switch to the folder that will be the parent for the new project folder. Type the following command:

```
cd <parent folder name>
```

  4. At the command line prompt, type:

```
compass create <the name of the project to be created>
```

Integrating Compass with IntelliJ IDEA

To develop a Compass-specific project in IntelliJ IDEA, you need to specify the Compass executable file `compass` and the project configuration file `config.rb` . You can do it either through the Compass Support page of the Settings dialog box or on the fly using an intention action that opens the Compass Support dialog box.

1. Open the Compass page or dialog box by doing one of the following:
   - Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Compass under Languages & Frameworks.
   - 1. In a `.sass` or `.scss` file, type the following import statement:

```
@import 'compass'
```

2. Click the red bulb icon or press `Alt+Enter`. Then choose Configure Compass from the suggestion list.

   The Compass Support dialog box opens.

2. To activate Compass support, select the Enable Compass support check box.
3. In the Compass executable file text box, specify the location of the `compass` executable file under the Ruby installation. Type the path manually, for example, `C:\Ruby200-x64\bin\compass`, or choose it from the drop-down list, or click the Browse button and choose the location of the `compass` file in the dialog box that opens.
4. In the Config path field, specify the location of the project Compass configuration file `config.rb`. Type the path manually, for example, `C:\my_projects\\compass_project\config.rb`, or choose it from the drop-down list, or click the Browse button and choose the location of the `compass` file int he dialog box that opens.
   The Compass configuration file `config.rb` is generated during project set-up through `compass create` or `compass init` commands.

Creating a Compass Sass or a Compass SCSS Transpiler

Sass and SCSS are not processed by browsers that work with CSS code. Therefore to be executed, Sass or SCSS code has to be translated into CSS. This operation is referred to as transpilation and the tools that perform it are called transpilers.

IntelliJ IDEA supports integration with a transpiler tool that translates Sass and SCSS code from a Compass project without changing the Compass-specific project structure.

In IntelliJ IDEA, transpiler configurations are called File Watchers. For each supported transpiler, IntelliJ IDEA provides a predefined File Watcher template. Predefined File Watcher templates are available at the IntelliJ IDEA level. To run a transpiler against your project files, you need to create a project-specific File Watcher based on the relevant template, at least, specify the path to the transpiler to use on your machine.

IntelliJ IDEA provides a common procedure and user interface for creating File Watchers of all types. The only difference is in the predefined templates you choose in each case.

1. Install and enable the File Watchers repository plugin.
   The plugin is not bundled with IntelliJ IDEA, but it is available from the IntelliJ IDEA plugin repository plugin repository. See Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins for details.
2. To start creating a File Watcher, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X on the main menu, and then click File Watchers under the Tools node. The File Watchers page that opens, shows the list of File Watchers that are already configured in the project.
3. Click the Add button or press `Alt+Insert` and choose the compass sass or compass scss predefined template from the pop-up list.
4. In the Program text box, specify the path to the executable file:
   - `compass.bat` for Windows
   - `compass` for Unix and OS X

   Type the path manually or click the Browse button and choose the file location in the dialog box that opens.
5. In the Arguments text box, type one of the following depending on the operating system used:
   - For OS X:
     - `compile $ProjectFileDir$` to process an entire directory
     - `compile $ProjectFileDir$ $FilePath$` to process a single file
   - For Windows:
     - `compile $UnixSeparators($ProjectFileDir$)$` to process an entire directory
     - `compile $UnixSeparators($FilePath$)$` to process a single file
   - For Linux (Ubuntu):
     - `compile $ProjectFileDir$` to process an entire directory
     - `compile $ProjectFileDir$ $FilePath$` to process a single file
6. Proceed as described on page Using File Watchers.

Running a Compass Sass or Compass SCSS Transpiler

When you open a Sass or SCSS file, IntelliJ IDEA checks whether an applicable file watcher is available in the current project. If such file watcher is configured but disabled, IntelliJ IDEA displays a pop-up window that informs you about the configured file watcher and suggests to enable it.

If an applicable file watcher is configured and enabled in the current project, IntelliJ IDEA starts it automatically upon the event specified in the New Watcher dialog.

- If the Immediate file synchronization check box is selected, the File Watcher is invoked as soon as any changes are made to the source code.
- If the Immediate file synchronization check box is cleared, the File Watcher is started upon save (File | Save All, `Ctrl+S`) or when you move focus from IntelliJ IDEA (upon frame deactivation).

IntelliJ IDEA creates a separate file with the generated output. The file has the name of the source Sass or SCSS file and the extension `css`. The location of the generated files is defined in the Output paths to refresh text box of the New Watcher dialog. However, in the Project Tree, they are shown under the source file which is now displayed as a node.

XML-Java Binding

In this section:

- XML-Java Binding
  - Introduction
  - Using JAXB and XmlBeans
- Generating Java Code from XML Schema
- Generating XML Schema from Java Code
- Generating Marshallers
- Generating Unmarshallers

Introduction

This section describes how to:

- Get a Java representation of an XML Schema
- Have an XML Schema generated on the basis of a Java class
- Unmarshal XML instance documents into Java content trees and vice versa.

With IntelliJ IDEA, this transformation can be done using one of the following data binders:

- Java Architecture for XML Binding (JAXB)
- XMLBeans

Using JAXB and XmlBeans

To use JAXB and XmlBeans, perform the following preliminary steps:

1. Download and install the XMLBeans tool.
2. On the Plugins page of the Settings/Preferences dialog box, make sure that the Web Services bundled plugin is enabled.
3. On the Web Services page of the Settings/Preferences dialog box, specify the installation directory of XmlBeans and one of the following Web Services engines:
   - Metro
   - GlassFish .
   - Java API for XML Web Services Reference Implementation (JAX-WS RI)
   - Java Web Services Development Pack (JWSDP)

Generating Java Code from XML Schema

On this page:

- Introduction
- Using JAXB
- Using XmlBeans

Introduction

This topic describes how to get a Java representation of an XML Schema , which involves mapping the elements of the XML Schema to members of a Java class. With IntelliJ IDEA, this transformation can be done using one of the following data binders:

- JAXB generates classes and groups them in Java packages. A package consists of a Java class name and an `ObjectFactory` class. The latter is a factory that is used to return instances of a bound Java class.
- XmlBeans converts an XML Schema into a Java class, compiles it, and places in the specified output `.jar` file.

Using JAXB

To generate a Java class from an XML Schema using JAXB

1. In the active editor tab, open the desired Schema (`.xsd`) file or an XML document which contains the desired Schema. Then choose Tools | JAXB | Generate Java Code From XML Schema Using JAXB on the main menu.

2. In the Generate Java from Xml Schema using JAXB dialog box that opens configure the generation procedure:
   - In the Schema/DTD/WSDL Path drop-down list, specify the file to be used as the basis for code generation. By default, the field shows the full path to the current file. Accept this suggestion or click the Browse button and select the desired file in the Select XML Schema File for JAXB Generation that opens.
   - From the Output Path drop-down list, select the module source directory to place the generated Java class in.
   - In the Package Prefix drop-down list, specify the package to include the generated stubs in.
   - Using the check boxes, configure additional options, such as generating annotation, setting the read-only status, downloading and installing additional libraries.

Using XmlBeans

To generate and compile a Java class from an XML Schema using XmlBeans

1. In the active editor tab, open the desired Schema (`.xsd`) file or an XML document which contains the desired Schema. Then choose Tools | XmlBeans | Generate Java Code From XML Schema Using XmlBeans on the main menu.

2. In the Generate Java Code From XML Schema using XmlBeans dialog box that opens configure the generation procedure:
   - In the Schema Path drop-down list, specify the file to be used as the basis for code generation. By default, the field shows the full path to the current file. Accept this suggestion or click the Browse button and select the desired file in the Select XML Schema /WSDL File for Generation dialog box that opens.
   - In the Output Path drop-down list, specify the name of the `.jar` file to place the generated and compiled Java code in. By default, IntelliJ IDEA suggests to create a new file `types.jar` . To overwrite an existing file, click the Browse button and choose the desired file in the Select XML Schema / Wsdl File for generation dialog box that opens.
   - To have missing libraries downloaded and installed automatically, select the Add necessary libraries in order for generated code compile and work check box.

Generating Xml Schema From Java Code

This topic describes how to have an XML Schema generated on the basis of a Java class, which involves mapping the members of the Java class to the elements of the XML Schema. With IntelliJ IDEA, this transformation can be done using the JAXB .

To generate an XML Schema from a Java class using JAXB

1. Open the Generate XML Schema From Java Using JAXB dialog box by doing one of the following:
   - In the Project tool window, select the name of the desired class and choose Web Services | Generate XML Schema From Java Using JAXB on the context menu.
   - With the desired class opened in the active editor tab, choose Tools | JAXB | Generate XML Schema From Java Using JAXB on the main menu.

2. Specify the method parameter and return types to be reflected in the generated Schema:
   - To have all the class methods involved, clear the Include parameter and return type of the following methods check box.
   - To select specific methods to be involved, select the Include parameter and return type of the following methods check box, then select the Add to JAXB generation check box next to the desired methods.

Generating Marshallers

WithIntelliJ IDEA, you can have marshal code stubs generated using JAXB or XMLBeans binding tools.

To generate marshal code using JAXB

1. Position the cursor where you need the marshaller to be generated.

2. Do one of the following:
   - On the main menu, choose Tools | JAXB | JAXB Client Code | Generate JAXB marshal code (Java object to XML)
   - On the context menu, choose WebServices | JAXB Client Code | Generate JAXB marshal code (Java object to XML)

To generate marshal code using XMLBeans

1. Position the cursor where you need the marshaller to be generated.

2. Do one of the following:
   - On the main menu, choose Tools | XMLBeans | XMLBeans Client Code | Generate XMLBeans marshal code (Java object to XML)
   - On the context menu, choose WebServices | XMLBeans Client Code | Generate XMLBeans marshal code (Java object to XML)

Generating Unmarshallers

With IntelliJ IDEA, you can have unmarshal code stubs generated using JAXB or XMLBeans binding tools.

To generate unmarshal code using JAXB

1. Position the cursor where you need the unmarshaller to be generated.

2. Do one of the following:
   - On the main menu, choose Tools | JAXB | JAXB Client Code | Generate JAXB unmarshal code (Java object from XML).
   - On the context menu, choose WebServices | JAXB Client Code | Generate JAXB unmarshal code (Java object from XML).

To generate unmarshal code using XMLBeans

1. Position the cursor where you need the unmarshaller to be generated.

2. Do one of the following:
   - On the main menu, choose Tools | XMLBeans | XMLBeans Client Code | Generate XMLBeans unmarshal code (Java object from XML).
   - On the context menu, choose WebServices | XMLBeans Client Code | Generate XMLBeans unmarshal code (Java object from XML).

Node.js

In this section:

- Node.js
  - Introduction
  - Before you start
  - Configuring Node.js in IntelliJ IDEA
- Configuring Node.js Interpreters
- Running and Debugging Node.js
- Running Nodeunit Tests
- Installing and Removing External Software Using Node Package Manager
- V8 CPU and Memory Profiling
- Using Pug (Jade) Template Engine
- Running NPM Scripts

Introduction

IntelliJ IDEA supports integration with the Node.js runtime environment thus enabling running, debugging, and unit testing of Node.js applications.

IntelliJ IDEA recognizes Node.js code and provides basic coding assistance and highlighting for it. To get guidance in Node development, see HowToNode.org .

Before you start

1. Download and install the Node.js runtime environment.
2. Install and enable the NodeJS plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The Node.js plugin introduces the following changes to the IntelliJ IDEA UI:

- Node.js page is added to the Settings dialog box.
- Run/debug configurations are added.

Configuring Node.js in IntelliJ IDEA

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks. The Node.js and NPM page opens.
2. In the Node Interpreter field, specify the local Node.js interpreter to use. Choose the interpreter from the drop-down list or click and choose the interpreter in the dialog box that opens.
   The term local Node.js interpreter denotes a Node.js installation on your computer. The term remote Node.js interpreter denotes a Node.js installation on a remote host or in a virtual environment set up in a Vagrant instance. On the Node.js and NPM page, you can specify only local interpreters. Remote interpreters are configured in the Configure Node.js Remote Interpreter Dialog dialog which can be accessed only from the Run/Debug Configuration: Node JS dialog. See Configuring Node.js Interpreters for details.
3. In the Code Assistance area, configure the Node.js core module sources if they are not configured yet.
   When developing a Node.js application it can be convenient to have code completion, reference resolution, validation, and debugging capabilities for Node core modules (`fs` , `path` , `http` , etc.). However, these modules are compiled into the Node.js binary. IntelliJ IDEA provides the ability to configure these sources as a JavaScript library and associate it with your project.
   - If the Node.js core module sources are not set up, IntelliJ IDEA displays a notification Node.js Core Library is not enabled with an Enable button. Click this button to have IntelliJ IDEA configure Node.js Core sources automatically.
     When the configuration is completed, IntelliJ IDEA displays information about the currently configured version, the notification Node.js Core Library is enabled, and adds two buttons: the Disable button and the Usage scope button.
   - If the library is set up, IntelliJ IDEA displays information about the currently configured version, the notification Node.js Core Library is enabled, and adds two buttons: the Disable button and the Usage scope button.
     - Click the Disable button to discard the configuration of the Node.js Core libraries in the current project.
     - Click the Usage scope button to associate the desired directories with libraries.
4. If necessary, configure the scope in which the Node.js Core sources are treated as libraries. Click the Usage scope button, and in the Usage Scope dialog box that opens, click the desired directories, and from the drop-down list select the newly configured Node.js core module sources library.
   The use of a library is enabled recursively, that is, if a library is associated with a folder it is automatically enabled in all the nested directories and files.

Configuring Node.js Interpreters

On this page:

- Introduction
- Configuring a local Node.js interpreter
- Configuring a remote Node.js interpreter on a host accessible through SSH connection
- Configuring a remote Node.js interpreter in a Vagrant environment instance
- Configuring a remote Node.js interpreter on a remote host accessible through SFTP
- Configuring mappings

Introduction

With IntelliJ IDEA, you can use local and remote Node.js interpreters.

The term local Node.js interpreter denotes a Node.js installation on your computer. The term remote Node.js interpreter denotes a Node.js installation on a remote host or in a virtual environment set up in a Vagrant instance.

You can access a remote interpreter in three ways:

- Using SSH credentials to access the host where the Node.js interpreter is installed.
- Through access to the corresponding Vagrant instance.
- According to a Server Access Configuration . This approach is also helpful if you are going to synchronize your project sources with the Web server on the target remote host.

Configuring a local Node.js interpreter

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
2. On the Node.js and NPM page that opens, click the ... button next to the Node Interpreter drop-down list.
3. In the Node.js Interpreters Dialog that opens with a list of all the currently configured interpreters, click + in the toolbar. In the dialog box that opens, choose Add Local on the context menu , then choose the local installation of Node.js.
   The path to the chosen interpreter is shown in Node.js Interpreters Dialog .

Configuring a remote Node.js interpreter on a host accessible through SSH connection

Before you start:

1. Configure access to an `ssh` server on the target remote host and make sure this server is running.
2. Make sure the Node.js Remote Interpreter repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

To configure a Node.js interpreter using SSH credentials:

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button +, and choose Node.js on the context menu. In the Run/Debug Configuration: Node JS dialog that opens, click ... next to the Node interpreter field.
2. In the Node.js Interpreters Dialog that opens with a list of all the currently configured interpreters, click + in the toolbar. In the dialog box that opens, choose Add Remote on the context menu .
3. In the Configure Node.js Remote Interpreter Dialog that opens, choose the SSH Credentials method.
4. Specify the name of the remote host and the port which the SSH server listens to. The default port number is 22.
5. Specify your credentials to access the remote host in accordance with the credentials received during the registration on the server. Type your user name and choose the authentication method:
   - To access the host through a password, choose Password from the Auth type drop-down list and type the password.
   - To access the host through a pair of SSH keys, choose Key pair, then specify the path to the file where your private key is stored or the passphrase if you have configured it during the generation of the key pair.
6. Specify the location of the Node.js executable file in accordance with the development environment configured on the selected remote host. By default IntelliJ IDEA suggests the `/usr/bin/node` folder. To specify another folder, click the Browse button ... and choose the relevant folder in the dialog box that opens. Note that the Node.js home directory must be open for edit.
7. When you click OK, IntelliJ IDEA checks whether the Node.js executable is actually stored in the specified folder.
   - If no Node.js executable is found, IntelliJ IDEA displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
   - If the Node.js executable is found, you return to the Node.js Interpreters where the installation folder and the detected version of the Node.js interpreter are displayed.

Configuring a remote Node.js interpreter in a Vagrant environment instance

Before you start:

1. Make sure that Vagrant and Oracle's VirtualBox are downloaded, installed, and configured on your computer as described in Vagrant .
2. Make sure the Vagrant and Node.js Remote Interpreter plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.
3. Make sure the Node.js Remote Interpreter repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
4. Make sure that the parent folders of the following executable files are added to the system PATH variable:
   - `vagrant.bat` or `vagrant` from your Vagrant installation. This should be done automatically by the Vagrant installer.
   - `VBoxManage.exe` or `VBoxManage` from your Oracle's VirtualBox installation.
5. Configure the Node.js development environment in the Vagrant instance to be used. Learn more about using Vagrant with IntelliJ IDEA in Vagrant .

To configure a Node.js Interpreter in a Vagrant instance

On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button + , and choose

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button, and choose Node.js on the context menu. In the Run/Debug Configuration: Node JS dialog that opens, click next to the Node interpreter field.
2. In the Node.js Interpreters Dialog that opens with a list of all the currently configured interpreters, click in the toolbar. In the dialog box that opens, choose Add Remote on the context menu .
3. In the Configure Node.js Remote Interpreter Dialog that opens, choose the Vagrant method.
4. Specify the Vagrant instance folder which points at the environment you are going to use. Technically, it is the folder where the VagrantFile configuration file for the desired environment is located. Based on this setting, IntelliJ IDEA detects the Vagrant host and shows it as a link in the Vagrant Host URL read-only field.
5. Specify the location of the Node.js executable file in accordance with the development environment configured in the selected Vagrant instance. By default IntelliJ IDEA suggests the `/usr/bin/node` folder. To specify another folder, click the Browse button and choose the relevant folder in the dialog box that opens. Note that the Node.js home directory must be open for edit.
6. When you click OK, IntelliJ IDEA checks whether the Node.js executable is actually stored in the specified folder.
   - If no Node.js executable is found, IntelliJ IDEA displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
   - If the Node.js executable is found, you return to the Node.js Interpreters where the installation folder and the detected version of the Node.js interpreter are displayed.

Configuring a remote Node.js interpreter on a remote host accessible through SFTP

Before you start:

1. Make sure a `ssh` server is running on the target remote host and you have configured access to it.
2. Make sure the Node.js Remote Interpreter repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Make sure the Remote Hosts Access plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .
4. Make sure you have at least one IntelliJ IDEA-wide server access configuration of the type SFTP to establish access to the target host. To make a configuration available in all IntelliJ IDEA projects, clear the Visible only for this project check box in the Deployment: Connection Tab . See Creating a Remote Server Configuration for details.

To configure a remote Node.js interpreter based on an SFTP server access configuration

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button, and choose Node.js on the context menu. In the Run/Debug Configuration: Node JS dialog that opens, click next to the Node interpreter field.
2. In the Node.js Interpreters Dialog that opens with a list of all the currently configured interpreters, click in the toolbar. In the dialog box that opens, choose Add Remote on the context menu .
3. In the Configure Node.js Remote Interpreter Dialog that opens, choose the Deployment Configuration method.
4. From the Deployment Configuration drop-down list, choose the server access configuration of the type SFTP according to which you want IntelliJ IDEA to connect to the target host. If the settings specified in the chosen configuration ensure successful connection, IntelliJ IDEA displays the URL address of the target host as a link in the Deployment Host URL field.
   To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . By default, IntelliJ IDEA retrieves path mappings from the chosen server access (deployment) configuration. If the configuration does not contain path mappings, IntelliJ IDEA displays the corresponding error message.

   To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the relevant server access configuration, switch to the Mappings tab, and map the local folders to the folders on the server as described in Creating a Remote Server Configuration , section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.
5. Specify the location of the Node.js executable file in accordance with the development environment configured on the selected remote host. By default IntelliJ IDEA suggests the `/usr/bin/node` folder. To specify another folder, click the Browse button and choose the relevant folder in the dialog box that opens. Note that the Node.js home directory must be open for edit.
6. When you click OK, IntelliJ IDEA checks whether the Node.js executable is actually stored in the specified folder.
   - If no Node.js executable is found, IntelliJ IDEA displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
   - If the Node.js executable is found, you return to the Node.js Interpreters where the installation folder and the detected version of the Node.js interpreter are displayed.

Configuring mappings

When you debug an application with a remote Node.js interpreter, the debugger tells IntelliJ IDEA the name of the currently processed file and the number of the line to be processed. IntelliJ IDEA opens the local copy of this file and indicates the line with the provided number. This behaviour is enabled by specifying correspondence between files and folders on the server and their local copies. This correspondence is called mapping, it is set in the debug configuration.

If you use an interpreter accessible through SFTP connection or located on a Vagrant instance, the mappings are automatically retrieved from the corresponding deployment configuration or `Vagrantfile` . To specify additional mappings:

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button, and choose Node.js on the context menu.
2. In the Run/Debug Configuration: Node JS dialog that opens, choose the required remote interpreter from the Node interpreter drop-down list.
3. Click next to the Path Mappings field.
4. The Edit Project Path Mappings Dialog that opens, shows the path mappings retrieved from the deployment configuration or `Vagrantfile` . These mappings are read-only.
   - To add a custom mapping, click and specify the path in the project and the corresponding path on the remote runtime environment in the Local Path and Remote Path fields respectively. Type the paths manually or click and select the relevant files or folders in the dialog box that opens.
   - To remove a custom mapping, select it in the list and click .

Running and Debugging Node.js

On this page:

- Before you start
- Local and remote modes of running or debugging Node.js applications
- Running a Node.js application
- Debugging a Node.js application locally
- Debugging a running Node.js application
- Creating a Node.js run/debug configuration
- Enabling Live Editing during a Node.js debugging session
- Creating a Node.js remote debug configuration.
- Node.js Multiprocess Debugging

Before you start

Install and enable the NodeJS plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Local and remote modes of running or debugging Node.js applications

Running a Node.js application in IntelliJ IDEA is supported only in the local mode. This means that IntelliJ IDEA itself starts the Node.js engine and the target application according to a run configuration and gets full control over the session.

Debugging can be performed in two modes:

- Locally, with the Node.js engine started from IntelliJ IDEA.
- Remotely, when IntelliJ IDEA connects to an already running Node.js application. This approach gives you the possibility to re-start a debugging session without re-starting the Node.js server.

In either case, the debugging session is initiated through a debug configuration

You can also configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, IntelliJ IDEA creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated NodeJS run configuration.

Running a Node.js application

1. Create a Node.js run configuration .
2. To launch the application, select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button . The Run tool window opens.
3. Open the browser of your choice and open the page with the URL address generated through the server.listen function based on the `port` and `host` parameters. The page shows the result of executing your Node.js application.

Debugging a Node.js application locally

1. Set the breakpoints in the Node.js code, where necessary. At least one breakpoint is necessary otherwise the program will be just executed. If you want the debugging tool to stop at the first line of your code, set a breakpoint at the first line.
2. Create a Node.js run/debug configuration .
3. To start a debugging session, select the required debug configuration from the list on the main tool bar and then choose Run | Debug <configuration name> on the main menu or click the Debug toolbar button .
4. Open the browser of your choice and open the starting page of your application. Control over the debugging session returns to IntelliJ IDEA.
5. Switch to IntelliJ IDEA, where the controls of the Debug tool window are now enabled. Proceed with the debugging session step through the breakpoints , switch between frames, change values on-the-fly, examine a suspended program , evaluate expressions , and set watches .

Debugging a running Node.js application

With IntelliJ IDEA, you can connect to an already running Node.js applications. The application can be started either on the same machine or on a physically remote host.

When the application to debug is running on a physically remote host, you need to run a proxy or any other software that ensures port forwarding on the Node.js server. This is necessary because the debug port can open only on the `localhost` network interface. The `localhost` network interface cannot be accessed from another machine therefore IntelliJ IDEA cannot connect to it upon initiating a debugging session.

1. Make sure the application to debug has been launched in the target environment with the following parameters: `--debug-brk=<port through which the debugger on the remote host interacts with the network interface which accepts external connections>`
   Please note the following:
   - With the `--debug-brk` option, the execution of the application suspends right after launch. This option allows you to debug the code executed on start.
   - With the `--debug` option, the code that has to be executed on the application start is executed whereupon the application waits for a debugger to connect to it. This option is useful when you are not going to debug Node.js right now, but you want to debug it later.
2. Create a Node.js Remote Debug configuration: in the Debug port text box, type the port number through which you will interact with the remote host according to the server access configuration, see Creating a Remote Server Configuration .
3. With the application still running, launch the Node.js Remote Debug configuration (select the configuration in the list and click the Debug toolbar button).
4. In the Run tool window, copy the URL address of the server and open the corresponding page in the browser. Control over the debugging session returns to IntelliJ IDEA.
5. Switch to IntelliJ IDEA. In the Debug tool window, step through the breakpoints , switch between frames, change values on-the-fly, examine a suspended program , evaluate expressions , and set watches .

Creating a Node.js run/debug configuration

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button , and choose Node.js on the context menu.

3. In the Run/Debug Configuration: Node.js dialog box, that opens, specify the following:
   - The name of the configuration.
   - In the Node Interpreter field, specify the Node.js installation to use. Choose the local or remote interpreter from the drop-down list, or click ... and choose the interpreter in the dialog box that opens, or configure an interpreter as described in Configuring Node.js Interpreters .
   - To enable remote debugging of the application, specify the following option in the Node parameters text box: `--debug-brk=<port for connect to debugger remotely>`
     - With the `--debug-brk` option, the execution of the application suspends right after launch. This option allows you to debug the code executed on start.
     - With the `--debug` option, the code that has to be executed on the application start is executed whereupon the application waits for a debugger to connect to it. This option is useful when you are not going to debug Node.js right now, but you want to debug it later.
   - In the JavaScript File field, specify the location of the file to start running the Node.js application from.
   - If the file to run references any other files, specify their location in the Working directory field.
   - If applicable, in the Application parameters text box, specify the arguments to be passed to the application on start through the process.argv array.
4. If necessary, configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, IntelliJ IDEA creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated NodeJS run configuration.
5. Click OK, when ready.

Enabling Live Editing during a Node.js debugging session

You can configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, IntelliJ IDEA creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated NodeJS run configuration.

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. From the list, choose the Node.js run configuration to activate the Live Edit functionality in. In the dialog box that opens, switch to the Browser / Live Edit tab.
3. Select the After launch check box to have a browser started automatically after a debugging session is launched. Specify the browser to use in the drop-down list next to the check box.
   - To use the system default browser, choose Default.
   - To use a custom browser, choose it from the list. Note that Live Edit is fully supported only in Chrome.
   - To configure browsers, click the Browse button ... and adjust the settings in the Web Browsers dialog box that opens. For more information, see Configuring Browsers .
4. Select the With JavaScript debugger check box to enable the JavaScript debugger in the selected browser and specify the URL address to open the application at.

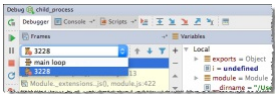
Creating a Node.js remote debug configuration.

1. On the main menu, choose Run | Edit Configurations.
2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button +, and choose Node.js Remote Debug on the context menu.
3. In the Run/Debug Configuration: Node.js Remote Debug dialog box, that opens, specify the following:
   - The name of the configuration.
   - The host where the target application is running.
   - The port to connect to. Copy the port number from the information message in the Run tool window that controls the running application.
4. Click OK, when ready.

Node.js Multiprocess Debugging

IntelliJ IDEA supports debugging additional Node.js processes that are launched by the child_process.fork() method or by the cluster module . Such processes are shown as threads in the Frame pane on the Debugger tab of the Debug Tool Window .

1. Set the breakpoints in the processes to debug.
2. Create a Node.js run/debug configuration.
3. Choose the newly created configuration in the Select run/debug configuration drop-down list on the tool bar and click the Debug toolbar button .
   The Debug tool window opens and the Frames drop-down list shows the additional processes as threads as soon as they are launched:



   To examine the data (variables, watches, etc.) for a process, select its thread in the list and view its data in the Variables and Debug Tool Window. Watches panes. When you select another process, the contents of the panes are updated accordingly.

Running Nodeunit Tests

On this page:

- Introduction
- Creating and running unit tests for Node.js applications
- Enabling unit testing for Node.js applications
- Creating Nodeunit tests
- Creating a Nodeunit run configuration

Introduction

IntelliJ IDEA supports integration with the nodeunit framework thus enabling running unit test for Node.js applications.

This topic provides guidelines in Node.js-specific unit testing procedures. For general information on testing in IntelliJ IDEA, see the section Testing .

Creating and running unit tests for Node.js applications

To create and run unit tests for a Node.js application, perform these general steps

1. Enable nodeunit support .
2. Write the unit tests .
3. Mark the folder where the unit tests are stored as a test source folder, see Configuring Content Roots .
4. Create a run configuration of the type Nodeunit.
5. Launch unit tests and monitor test results in the Run tool window.

Enabling unit testing for Node.js applications

To enable unit testing for Node.js applications

1. Download, install, and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the Node.js runtime environment.
3. Download and install the nodeunit testing framework

Creating Nodeunit tests

To create Nodeunit tests

- Create a folder `test` at the same level as the `src` folder
- Populate the `test` folder. For each production file, create a separate test file.
- Mark the folder where the unit tests are stored as a test source folder, see Configuring Content Roots .

Creating a Nodeunit run configuration

To create a Nodeunit run configuration

1. Open the Run/Debug Configuration dialog box by choosing Run | Edit Configurations on the main menu.
2. Click the Add button + on the toolbar and select the Nodeunit configuration type.
3. In the dialog box that opens, specify the following:
   1. The name to identify the configuration.
   2. The path to the NodeJS installation to use.
      If you have appointed one of the installations as default , the field displays the path to its executable file.
   3. The working directory. This can be the project root folder or the parent directory for the `test` folder.
   4. The scope of tests to run.
      - To have IntelliJ IDEA run all the test files in a folder, choose All JavaScript test files in the directory from the Run drop-down list. In the Directory text box, type the path to the test folder relative to the working directory .
      - To have a specific test executed, choose JavaScript test file from the Run drop-down list. In the JavaScript test file text box, type the path to the file relative to the working directory .
4. Apply the changes and close the dialog box.

Installing and Removing External Software Using Node Package Manager

On this page:

- Introduction
- Installing Node.js and Node Package Manager (npm)
- Installing an external tool globally
- Installing an external tool in a project
- Installing an external tool as a development dependency

Introduction

A number of tools are started through Node.js, for example, the CoffeeScript, TypeScript, and Less transpilers, YUI, UglifyJS, and Closure compressors, Karma test runner, Grunt task runner, etc. The Node Package Manager (npm) is the easiest way to install these tools, the more so that you have to install Node.js anyway.

Depending on the desired location of the tool executable file, choose one of the following methods:

- Install the tool globally at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project.
- Install the tool in a specific project and thus restrict its use to this project.
- Install the tool in a project as a development dependency.

In either installation mode, make sure that the parent folder of the tool is added to the `PATH` variable. This enables you to launch the tool from any folder.

Installing Node.js and Node Package Manager (npm)

1. Download and install Node.js. The runtime environment is required for two reasons:
   - The tool is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the tool.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the tool and npm from any folder.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins.

Installing an external tool globally

Global installation makes a tool available at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project. Moreover, during installation the parent folder of the tool is automatically added to the `PATH` variable, which enables you to launch the tool from any folder. To install the tool globally, do one of the following:

- Run the installation from the command line in the global mode:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs.
  3. Type the following command at the command line prompt:

     ```
     npm install -g  <tool name>
     ```

     The `-g` key makes the tool run in the global mode. Because the installation is performed through NPM, the tool is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the tool from any folder.

     For more details on the NPM operation modes, see npm documentation. For more information about installing the tool, see https://npmjs.org/package/.
- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click .
  3. In the Available Packages dialog box that opens, select the required package to install.
  4. Select the Options check box and type `-g` in the text box next to it.
  5. Optionally specify the product version and click Install Package to start installation.

Installing an external tool in a project

Installing a tool in a specific project restricts its use to this project. To run project installation, do one of the following:

- Run the installation from the command line:
  1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
  2. Switch to the project root folder and type the following command at the command line prompt:

     ```
     npm install  <tool name>
     ```
- Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
  2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click .

3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in template-based projects of the type Node Boilerplate or Node.js Express, which already have the `node_modules` folder. The latter is important because NPM installs the tool in a `node_modules` folder. If your project already contains such folder, the tool is installed there.

Projects of other types or empty projects may not have a `node_modules` folder. In this case npm goes upwards in the folder tree and installs the tool in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be outside your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the tool is installed there.

In either case, make sure that the parent folder of the tool is added to the `PATH` variable. This enables you to launch the tool from any folder.

Installing an external tool as a development dependency

If a tool is a documentation or a test framework, which are of no need for those who are going to re-use your application, it is helpful to have it excluded from download for the future. This is done by marking the tool as a development dependency , which actually means adding the tool in the `devDependencies` section of the `package.json` file.

With IntelliJ IDEA, you can have a tool marked as a development dependency right during installation. Do one of the following:

– Run the installation from the command line:

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install --dev  <tool name>
```

– Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the package.
4. Select the Options check box and type `--dev` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

After installation, a tool is added to the `devDependencies` section of the `package.json` file.

V8 CPU and Memory Profiling

On this page:

- Introduction
- Why is profiling important
- Before you start
- Preparing for V8 CPU and memory heap profiling
  - Installing the v8-profiler Package
- CPU profiling
  - Configuring CPU Profiling
  - Collecting CPU Profiling Information
  - Analyzing CPU Profiling Information
    - Exploring Call Tress
    - Analyzing the Flame Chart
      - Selecting a Fragment in the Timeline
      - Synchronization in the Flame Chart
- Memory profiling
  - Configuring Memory Profiling
  - Collecting Memory Profiling Information
  - Analyzing Memory Profiling Information
    - Navigating through a Snapshot

Introduction

As you may know, V8 is an open-source JavaScript engine developed by Google. It is used in Google Chrome, Chromium , Node.js, and io.js .

V8 has a sampling CPU profiler V8 profiler which is intended for capturing and analyzing CPU profiles and heap snapshots for your Node.js applications. With V8 CPU profiling you can get a better understanding of which parts of your code take up the most CPU time, and how your code is executed and optimized by the V8 JavaScript engine.

You can also open and explore profiles and snapshots captured in Google Chrome DevTools for your client-side code.

Why is profiling important

- A carefully designed algorithm can make your code faster and manage your memory consumption better, even more efficiently than the virtual machine can. Profiling is the way to look inside the execution of your code and prove your assumptions about your design decisions.
- Profiling is especially relevant for Javascript, which is a powerful language with advanced features like dynamic typing, closures, and even the ability to create code at runtime. Therefore the behavior of the JavaScript engine is quite sophisticated, and there are cases when you really need to go deep into the details of how the engine works. Some of the code patterns you use need being tweaked to allow the JavaScript optimizer to do its work.
- JavaScript is by no means a simple language to judge if your code manages memory well. Closure memory leaks are a good illustration of the fact that code that looks good and simple may still cause leaks.

Before you start

1. Install the Node.js runtime environment version 0.11.0 or higher.
2. Install and enable the NodeJS plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Restart IntelliJ IDEA for the changes to take effect.

Preparing for V8 CPU and memory heap profiling

V8 CPU profiling is provided through the IntelliJ IDEA built-in functionality, you do not need to install any additional software.

V8 memory heap profiling is provided through the v8-profiler package.

The v8-profiler package can be installed from IntelliJ IDEA using the Node Package Manager (npm).

Installing the v8-profiler Package

You can install the v8-profiler package using the Node Package Manager (npm) either in the command line mode or on the Node.js and NPM page of the Settings dialog box.

To install the v8-profiler package globally, do one of the following:

- In the command line mode, type `npm install -g v8-profiler` .
- Use the Node.js and NPM page of the Settings dialog box:
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
  2. On the Node.js and NPM page that opens, click Install + in the Packages area.
  3. In the Available Packages dialog box that opens, select the v8-profiler package from the list, possibly using the search field.
  4. Select the Options check box and type `-g` in the text box next to it to have the package installed globally and thus make it available from any IntelliJ IDEA project.
  5. Click Install.
  6. When the package is installed successfully, close the dialog box. IntelliJ IDEA brings you back to the Node.js and NPM page where the package is added to the list.

Learn more about installing packages through NPM at Installing and Removing External Software Using Node Package Manager .

CPU profiling

With V8 CPU profiling you can get a better understanding of which parts of your code take up the most CPU time, and how your code is executed and optimized by the V8 JavaScript engine.

To identify the processes that consume most of your CPU, you can use two methods: sampling and tracing.

- When the sampling method is applied, you periodically record stack traces of your application. The periods between records are measured in conventional units referred to as ticks.
  This method does not guarantee very good accuracy or precision for the following reason: snapshots are taken at random moments therefore any function can happen to be recorded in a snapshot. However, sampling can give us a rough picture of where the most of time is spent.
- When the tracing method is used, we actively record tracing information by ourselves, directly in the code. It is obviously better to get exact measurements of how much time each method took, and also allows to count how many times the traced method was called. The disadvantage of this method is that it comes with bigger result distortion compared to sampling.

Result Distortion. Both sampling and tracing introduce delays into execution and therefore influence the profiling results. With sampling, delays can be estimated as some fixed amount of time for each sampling event and do not introduce greater distortion than the sampling method itself (i.e. the delay is much shorter than the sampling interval). With tracing, the profiling delay depends on the code and the places where we made tracing measurements. For instance, if a traced method is called inside other traced methods numerously, all inner delays will accumulate for the outer method. If so, it may be difficult to separate the execution time from tracing distortion.

Usually we use sampling and tracing methods together. We start with sampling to get an idea of which parts of our code take the most time, and then instrument the code with tracing calls to zero on the issues.

Measurements are made not only for the work of your code, but also activities performed by the engine itself, such as compilation, calls of system libraries, optimization, and garbage collection. The following time metrics are made for execution of functions themselves and for performing activities:

- Total: the number of ticks (the time) during which a function was executed or an activity was performed.
- Total%: the ratio of a function/activity execution time to the entire time when measurements were made.
- Self: the pure execution time of a function/activity itself, without the time spent on executing functions called by it.
- Self%: the ratio of the pure execution time of a function/activity to the entire time when the measurements were made.
- Of Parent: the ratio of the pure execution time of a function to the execution time of the function that called it (Parent).

Configuring CPU Profiling

To invoke V8 CPU profiling on application start, you need to specify additional settings in the Node.js run configuration according to which the application will be launched.

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. From the list, choose the Node.js run configuration to activate CPU Profiling in or create a new configuration as described in Running and Debugging Node.js .
3. Switch to the V8 Profiling pane and specify the following:
   1. Select the Record CPU profiling info check box.
   2. In the Log folder field, specify the folder to store recorded logs in. Profiling data are stored in V8 log files `isolate-<session number>` .

Collecting CPU Profiling Information

1. Select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button .
2. When the scenario that you need to profile is executed, stop the process by clicking on the tool bar of the Run tool window.

V8 log file will be processed by V8 scripts to calculate averaged call traces. IntelliJ IDEA opens the V8 Profiling Tool Window .

Analyzing CPU Profiling Information

Analyzing the profiling logs is available after the process stops because currently stopping and restarting profiling during execution of an application is not supported.

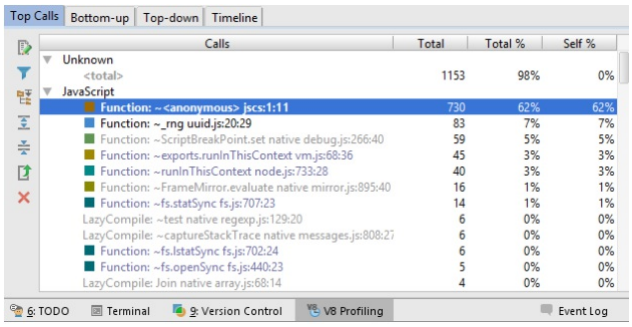
The collected profiling data is displayed in the V8 Profiling Tool Window which IntelliJ IDEA opens automatically when you stop your application. If the window is already opened and shows the profiling data for another session, a new tab is added. Tabs that were opened automatically are named after the run configurations that control execution of the applications and collecting the profiling data.

If you want to open and analyze some previously saved profiling data, choose V8 Profiling - Analyze V8 Profiling Log on the main menu and select the relevant V8 log file `isolate-<session number>` . IntelliJ IDEA creates a separate tab with the name of the log file.

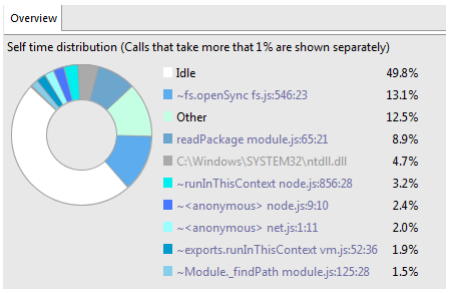
Exploring Call Tress

Based on the collected profiling data, IntelliJ IDEA builds three call trees and displays each of them in a separate pane. Having several call trees provides the possibility to analyze the application execution from two different points of view: on the one hand, which calls were time consuming ("heavy"), and on the other hand, "who called whom".

- The Top Calls pane shows a list of performed activities sorted in the descending order by the Self metrics. For each activity IntelliJ IDEA displays its Total, Total%, and Self% metrics. For each function call, IntelliJ IDEA displays the name of the file, the line, and the column where the function is defined.

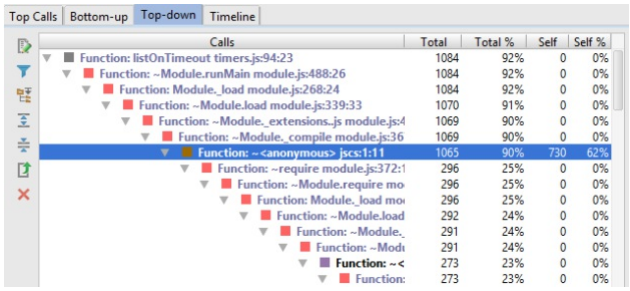


The diagram in the Overview pane shows distribution of self time for calls with the Self% metrics above 1%.



- The Bottom-up pane also shows the performed activities sorted in the descending order by the Self metrics. Unlike the Top Calls pane, the Bottom-up pane shows only the activities with the Total% metrics above 2 and the functions that called them. This is helpful if you encounter a heavy function and want to find out where it was called from.
  For each activity IntelliJ IDEA displays its execution time in ticks and the Of Parent metrics. For each function call, IntelliJ IDEA displays the name of the file, the line, and the column where the function is defined.

- The Top-down pane shows the entire call hierarchy with the functions that are execution entry points at the top. For each activity IntelliJ IDEA displays its Total, Total%, Self, and Self% metrics. For each function call, IntelliJ IDEA displays the name of the file, the line, and the column where the function is defined. Some of the functions may have been optimized by V8, see Optimizing for V8 for details.
  - The functions that have been optimized are marked with an asterisk (`*`) before the function name.
  - The functions that possibly require optimization but still have not been optimized are marked with a tilde (`~`) character before the function name. Though optimization may be delayed by the engine or skipped if the code is short-running, a tilde (`~`) points at a place where the code can be rewritten to achieve better performance.



- To navigate to the source code of a function, select the function in question in the tree and click on the toolbar or choose Jump to source on the context menu of the selection. The file with the source code of the selected function is opened in the editor with the cursor positioned at the function.

- When a tab for a profiling session is opened, by default the nodes with heaviest calls are expanded. While exploring the trees, you may like to fold some nodes or expand other ones. To restore the original tree presentation, click the Expand Heavy Traces button on the toolbar.

- To have IntelliJ IDEA display only the calls that indeed cause performance problems, filter out light calls:
  1. Click the Filter button on the toolbar.
  2. Using the slider, specify the minimum Total% or Parent% value for a call to be displayed and click Done.

- To expand or collapse all the nodes in the active pane, click or on the toolbar respectively.

- To expand or collapse a node, select it and choose Expand Node or Collapse Node on the context menu of the selection.

- Save and compare calls and lines:
  - To save a line with a function and its metrics, select the function and choose Copy on the context menu of the selection. This may be helpful if you want to compare the measurements for a function from two sessions, for example, after you make some improvements to the code.
  - To save only the function name and the name of the file where the function is defined, select the function and choose Copy Call on the context menu of the selection.
  - To compare an item with the contents of the Clipboard, select the item in question and choose Compare With Clipboard on the context menu of the selection. Compare the items in the Difference Viewer that opens.

- To save the call tree in the current pane to a text file, click on the toolbar and specify the target file in the dialog box that opens.

Analyzing the Flame Chart

Use the multicolor chart in the Flame Chart tab to find where the application paused and explore the calls that provoked these pauses. The chart consists of four areas:

- The upper area shows a timeline with two sliders to limit the beginning and the end of a fragment to investigate.
- The bottom area shows a stack of calls in the form of a multicolor chart. When called for the first time, each function is assigned a random color, whereupon every call of

this function within the current session is shown in this color.

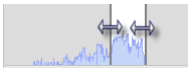
- The middle area shows a summary of calls from the Garbage Collector, the engine, the external calls, and the execution itself. The colors reserved for the Garbage Collector, the engine , the external calls, and the execution are listed on top of the area:

- The right-hand pane lists the calls within a selected fragment, for each call the list shows its duration, the name of the called function, and file where the function is defined.

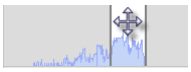
Selecting a Fragment in the Timeline

To explore the processes within a certain period of time, you need to select the fragment in question. You can do it in two ways:

- Use the sliders:



- Click the window between two sliders and drag it to the required fragment:



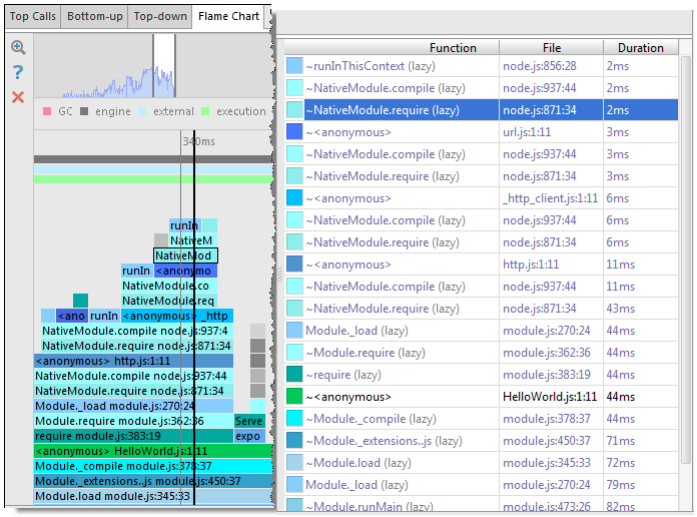
In either case, the multicolor chart below shows the stack of calls within the selected fragment.

To enlarge the chart, click the selected fragment and then click the Zoom button on the toolbar. IntelliJ IDEA opens a new tab and shows the selected fragment enlarged to fit the tab width so you can examine the fragment with more details.

Synchronization in the Flame Chart

The bottom and the right-hand areas are synchronized: as you drag the slider in the bottom area through the timeline the focus in the right-hand pane moves to the call that was performed at each moment.

Moreover, if you click a call in the bottom area, the slider moves to it automatically and the focus in the right-hand pane switches to the corresponding function, if necessary the list scrolls automatically. And vice versa, if you click an item in the list, IntelliJ IDEA selects the corresponding call in the bottom area and drags the slider to it automatically:



IntelliJ IDEA supports navigation from the right-hand area to the source code of called functions, to the other panes of the tool window, and to areas in the flame chart with specific metrics.

- To jump to the source code of a called function, select the call in question and choose Jump to Source on the context menu of the selection.
- To switch to another pane, select the call in question, choose Navigate To on the context menu of the selection, and then choose the destination:
  - Navigate in Top Calls
  - Navigate in Bottom-up
  - Navigate in Top-down

  IntelliJ IDEA switches to the selected pane and moves the focus to the call in question.
- To have the flame chart zoomed at the fragments with specific metrics of a call, select the call in question, choose Navigate To on the context menu of the slection, and then choose the metrics:
  - Navigate to Longest Time
  - Navigate to Typical Time
  - Navigate to Longest Self Time
  - Navigate to Typical Self Time

You can also navigate to the stacktrace of a call to view and anlyze exceptions. To do that, select the call in question and choose Show As Stacktrace. IntelliJ IDEA opens the stacktrace in a separate tab, to return to the Flame Chart pane, click V8 CPU Profiling tool window button in the bottom tool window.

Memory profiling

Though the V8 Javascript engine does memory management for you, memory leaks or dynamic memory problems are still possible. Here are some examples of memory leaks or shortcomings reasons:

- Using global objects to store collections of data, with complicated free policies.
- Errors in usages of closures: closures keep references onto outside objects.
- Keeping detached DOM nodes in javascript variables.

– Too frequent memory allocation.

Memory management control is especially important for Node.js applications, because the server-side code tend to run long while memory inaccuracy is accumulated.

V8 heap snapshots are complicated by their nature; they include many “engine” objects; the inner structure of objects differs from what you expect while reading your code. Learn more at Javascript Memory Profiling .

Configuring Memory Profiling

To allow taking memory snapshots, you need to specify additional settings in the Node.js run configuration according to which the application will be launched.

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. From the list, choose the Node.js run configuration to activate CPU Profiling in or create a new configuration as described in Running and Debugging Node.js .
3. Switch to the V8 Profiling pane and specify the following:
   1. Select the Allow taling heap snapshots check box.
   2. In the Log folder field, specify the folder to store recorded logs in. Profiling data are stored in V8 log files `isolate-<session number>` .
   3. Specify the v8-profiler package to use. Choose the relevant package from the v8-profiler package drop-down list or click the button next to it and choose the package in the dialog box that opens.
   4. Specify the port through which IntelliJ IDEA communicates with the profiler, namely, sends a command to take a snapshot when you click the Take Heap Snapshot button on the toolbar of the Run tool window.

Collecting Memory Profiling Information

1. Select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button .
2. At any time during the application execution, click the Take Heap Snapshot button on the toolbar of the Run tool window.
3. In the dialog box that opens, choose the folder to store the taken snapshot in and specify the name to save the snapshot file with. To start analyzing the snapshot immediately, select the Open snapshot check box.
4. Click OK to save the snapshot.

Analyzing Memory Profiling Information

The collected profiling data is displayed in the V8 Heap Tool Window which opens when you take a snapshot at choose to open it. If the window is already opened and shows the profiling data for another session, a new tab is added. Tabs that were opened automatically are named after the run configurations that control execution of the applications and collecting the profiling data.

If you want to open and analyze some previously saved mempry profiling data, choose V8 Profiling - Analyze V8 Heap Snapshot on the main menu and select the relevant `.snapshot` file. IntelliJ IDEA creates a separate tab with the name of the selected file.

The tool window has three tabs that present the collected information from difference point of views.

– The Containment tab shows the objects in you application grouped under several top-level entries: DOMWindow objects, Native browser objects, and GC Roots, which are roots the Garbage Collector actually uses. See Containment View for details.
  For each object, the tab shows its distance from the GC root, that is the shortest simple path of nodes between the object and the GC root, the shallow size of the object, and the retained size of the object. Besides the absolute values of the object's size, IntelliJ IDEA shows the percentage of memory the object occupies.
– The Biggest Objects tab shows the most memory-consuming objects sorted by their retained sizes . In this tab, you can spot memory leaks provoked by accumulating data in some global object.
– The Summary tab shows the objects in your application grouped by their types. The tab shows the number of objects of each type, their size, and the percentage of memory that they occupy. This information may be a clue to the memory state.

Each tab has a Details pane, which shows the path to the currently selected object from GC roots and the list of object’s retainers, that is, the objects that keep links to the selected object. Every heap snapshot has many “back” references and loops, so there are always many retainers for each object.

Navigating through a Snapshot

– To help differentiate objects and move from one to another without losing the context, mark objects with text labels. To set a label to an object, select the object of interest and click on the toolbar or choose Mark on the context menu of the selection. Then type the label to mark the object with in the dialog box that opens.

– To navigate to the function or variable that corresponds to an object, select the object of interest and click on the toolbar or choose Edit Source on the context menu of the selection. If the button and the menu option are disabled, this means that IntelliJ IDEA has not found a function or a variable that corresponds to the selected object.
  If several functions or variables are found, they are shown in a pop-up suggestion list.

– To jump from an object in the Biggest Objects or Summary tab or Occurrences view to the same object in the Containment tab, select the object in question in the Biggest Objects or Summary tab and click on the toolbar or choose Navigate in Main Tree on the context menu of the selection. This helps you investigate the object from the containment point of view and concentrate on the links between objects.

– To search through a snapshot:
  1. In the Containment tab, click on the toolbar.
  2. In the V8 Heap Search Dialog that opens, specify the search pattern and the scope to search in. The available scopes are:
     – Everywhere: select this check box to search in all the scopes. When this check box is selected, all the other search types are disabled.
     – Link Names: select this check box to search among the object names that V8 creates when calling the C++ runtime, see http://stackoverflow.com/questions/11202824/what-is-in-javascript .
       In the V8 Heap Tool Window , link names are marked with the `%` character (`%<link name>`).

- Class Names: select this check box to search among functions-constructors.
- Text Strings: select this check box to perform a textual search in the contents of the objects.
- Snapshot Object IDs: select this check box to search among the unique identifiers of objects. V8 assigns such a unique identifier in the format to each object when the object is created and preserves it until the object is destroyed. This means that you can find and compare the same objects in several snapshots taken within the same session.
  In the V8 Heap Tool Window , object IDs are marked with the `@` character (`@<object id>`).
- Marks: select this check box to search among the labels you set to objects manually by clicking on the toolbar of the Containment tab.

The search results are displayed in the Details pane, in a separate Occurrences of '<search pattern>' view. To have the search results shown grouped by the search scopes you specified, press the Group by Type toggle button on the toolbar.

When you open the dialog box next time, it will show the settings from the previous search.

Using Pug (Jade) Template Engine

IntelliJ IDEA supports integration with the Pug (Jade) template engine.

On this page:

- Before you start
- Changes to the UI
- Using Pug(Jade) templates in a NodeJS application

Before you start

1. Enable Node.js support as described in Node.js .
2. Make sure the NodeJS and Pug (ex-Jade) plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Also, if you need a file watcher, make sure that the plugin File Watchers is installed and enabled.

The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Changes to the UI

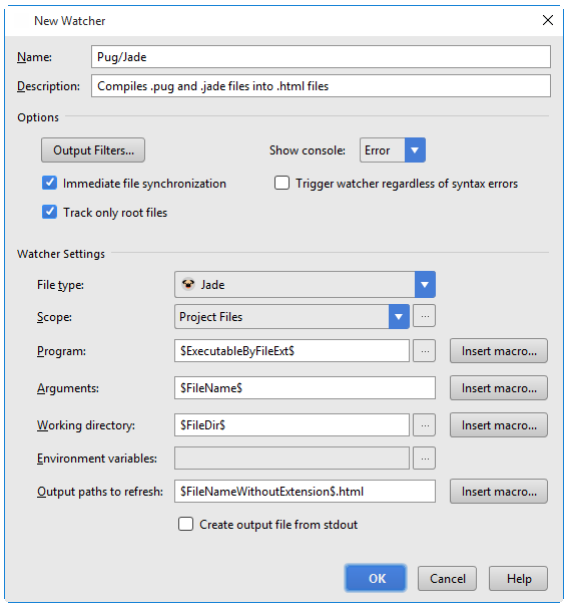
The Pug (ex-Jade) plugin introduces the following changes to the IntelliJ IDEA UI:

- The Jade file item is added to the New menu.
- The Pug files are marked with the icon ; the Jade files are marked with the icon .
- Coding assistance is provided in the Pug (Jade)-specific and HTML context:
  - Code formatting
  - Syntax highlighting
  - Code completion
  - Color schemes

Using Pug(Jade) templates in a NodeJS application

At runtime, the Pug (Jade) files fill be transformed into HTML pages.

1. Create a project from scratch, or around existing sources, or based on a NodeExpress template. See Creating a Project from Scratch , Generating a Project from a Framework Template for details.
2. Create a Pug (Jade) file. Follow these steps:
   1. In the Project tool window , select the directory in which you want to create a new file. To do that, for example, choose File | New.
   2. On the context menu, choose Jade file and specify the file name in the dialog box that opens.
3. Create a File Watcher to transform files with the extension `.jade` or `.pug` into `.html` pages:
   1. Click the Add Watcher link in the upper right-hand corner of the editor.
   2. In the New Watcher Dialog , accept the default predefined settings.
      Note that if the executable is in the PATH, then you should not specify it explicitly. Depending on the file extension (`.jade` or `.pug`), the corresponding executable is invoked.



4. As you edit a `.pug` / `.jade` file, IntelliJ IDEA invokes the file watcher which creates an `.html` file with the name of the processed `.pug` / `.jade` file and stores the generated `html` code in it.

See Using File Watchers for details.

Running NPM Scripts

On this page:

- Introduction
- Before you start
- Building a tree of scripts
- Running npm scripts from the tree of scripts
- Running tasks according to a run configuration
- Running npm scripts automatically
- Running a script as a as a before-launch task

Introduction

IntelliJ IDEA provides interface for running npm scripts . npm scripts support involves:

- Parsing `package.json` files, recognizing definitions of scripts.
- Building trees of scripts.
- Navigation between a script in the tree and its definition in the `package.json` file.
- Running and debugging scripts.
- Configuring the script execution mode and output.

Before you start

1. Download and install Node.js which contains npm .
2. Install and enable the NodeJS plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Create a `package.json` file with the `scripts` property with the definitions of the scripts to run.
4. To enable debugging a script, add the `$NODE_DEBUG_OPTION` to its definition in the `package.json` file, for example:

```
{
                    "name": "application-name",
                    "version": "0.0.1",
                    "scripts": {
                    "main": "node $NODE_DEBUG_OPTION ./app-compiled.js"
                    }
                    }
```

Scripts are launched in four ways:

- From a tree of scripts in the dedicated NPM Tool Window . The tool window opens when you invoke npm by choosing Show npm Scripts on the context menu of a `package.json` in the Project tool window or of a `package.json` opened in the editor.
- According to a dedicated run configuration, see Run/Debug Configuration: NPM .
- Automatically, as a start-up task.
- As a before-launch task, from another run configuration.

The result of executing a script is displayed in the Run tool window . The tool window shows the npm script output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed script is displayed on the title bar of the tool window.

Building a tree of scripts

- If the npm tool window is not opened yet, do one of the following:
  - Select the required `package.json` file in the Project tool window and choose Show npm Scripts on the context menu of the selection.
  - Open the required `package.json` file in the editor and choose Show npm Scripts on the context menu of the editor.

  In either case, the npm tool window opens showing the scripts tree built according to the selected or opened `package.json` file.
- If the npm tool window is already opened, click on the toolbar and choose the required `package.json` file from the list. IntelliJ IDEA adds a new node and builds a scripts tree under it. The title of the node shows the path to the `package.json` file according to which the tree is built.
- To re-build a tree, switch to the required node and click on the toolbar.

Running npm scripts from the tree of scripts

- To run a script, do one of the following:
  - Double click the required script in the tree.
  - Select the required script and choose Run <script name> on the context menu of the selection.
  - Select the required script and press `Enter`.
- To run several scripts, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required scripts, then choose Run on the context menu of the selection.

Running tasks according to a run configuration

Besides using temporary run configurations that IntelliJ IDEA creates automatically, you can create and launch your own npm run configurations. For details about run configurations, see Run/Debug Configuration and Creating and Editing Run/Debug Configurations .

To create an npm run configuration:

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. Click the Add New Configuration button on the toolbar and select npm from the pop-up list.

3. In the Run/Debug Configuration: NPM dialog box that opens, specify the name of the run configuration, the npm command line command to execute, the scripts to run (use blank spaces as separators), the location of the `package.json` file to retrieve the definitions of the scripts from, and the command line arguments to execute the script with.
   Specify the location of the Node executable file and the NodeJS-specific options to be passed to this executable file, see Node parameters for details.

   If applicable, specify the environment variables for the NodeJS executable file.

To run a script according to a run configuration, select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button. The output is displayed in the Run tool window .

Running npm scripts automatically

If you have some scripts that you run on a regular basis, you can add the corresponding run configurations to a list of startup tasks. The tasks will be executed automatically on the project start-up.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Startup Tasks under Tools.
2. On the Startup Tasks page that opens, click + on the toolbar.
3. From the drop-down list, choose the required npm run configuration. The configuration is added to the list.
   If no applicable configuration is available in the project, click + and choose Edit Configurations. Then define a configuration with the required settings in the Run/Debug Configuration: NPM page that opens. When you save the new configuration it is automatically added to the list of startup tasks.

Running a script as a as a before-launch task

1. Open the Run/Debug Configurations dialog by choosing Run | Edit Configurations on the main menu, and select the required configuration from the list or create it anew by clicking + and choosing the relevant run configuration type.
2. In the dialog box that opens, click + in the Before launch area and choose Run npm script from the drop-down list.
3. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
   Specify the location of the Node.js interpreter and the parameters to pass to it.

OSGi and OSMORC

In this section:

- OSGi and OSMORC
  - Introduction
  - Prerequisite
  - Creating a project or module with OSGi support
  - Adding OSGi support to the existing module
- Importing a Project from Bnd/Bndtools Model
- Set up a New Project
- Set up
- Settings
- Run Configurations

Introduction

IntelliJ IDEA lets you create OSGi-based applications using Osmorc plugin. IntelliJ IDEA also lets you import Bnd/Bndtools projects . You can create a project with the OSGi support, add a module with the OSGI support to the existing project, or add the OSGi support to an existing module.

Prerequisite

Before you start working with OSGi, make sure that the Osmorc plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Creating a project or module with OSGi support

To create a project or module with OSGi support

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Java. In the right-hand part of the page, specify Project SDK and select OSGi from the list. In Libraries area specify OSGi library settings:
   - Use library - select this option to specify the existing OSGi library.
   - Download - select this option to download the library. You can click Configure to edit downloading options.
   - Set up library later - select this option to configure the library later in your project.

   Click Next.

3. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Adding OSGi support to the existing module

To add OSGi support to the existing module

1. In the project tree right-click the module to which you want to add the OSGi support.

2. From the drop-down list select Add Framework Support.

3. In the dialog that opens,from the left-hand pane, select OSGi.

4. On the right side of the page, in the Libraries area, specify OSGi library settings.

   The OSGI library settings are as follows:
   - Use library - select this option to specify the existing OSGi library.
   - Download - select this option to download the library. You can click Configure to edit downloading options.
   - Set up library later - select this option to configure the library later in your project.

Importing a Project from Bnd/Bndtools Model

IntelliJ IDEA lets you import Bnd/Bndtools projects.

Before import your project, make sure that Osmorc plugin is enabled in IntelliJ IDEA. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not enabled, enable the plugin .

To import an external Bnd/Bndtools model into IntelliJ IDEA, follow these steps

1. If no project is currently open in IntelliJ IDEA, click Import Project on the Welcome screen . Otherwise, select File | New | Project from Existing Sources.
2. In the dialog that opens , select the directory that contains the project to be imported, or a file that contains an appropriate project description. Click OK.
3. On the first page of the Import Project wizard, select Bnd/Bndtools, and click Next. (This page is not shown if IntelliJ IDEA has guessed what you are importing.)
4. On the next page of the wizard, specify Bnd/Bndtools projects you want to import and click Next.
5. On the next page of the wizard, specify project SDK and click Finish.

Set Up a New Project

This topic describes how to set up a new project with Osmorc. Please take a look at Importing an Eclipse Workspace , if you want to import an existing Eclipse workspace intoIntelliJ IDEA.

From this section you will learn how to activate Osmorc for a module. Nothing about the creation of projects and modules which will use Osmorc is Osmorc-specific. So create them just like you would create any other IntelliJ IDEA projects and modules.

Osmorc is activated in a module via the dedicated Osmorc facet .

To add the Osmorc facet to a module, do one of the following:

- Add the facet explicitly .
- Have the facet detected automatically.

To add a facet explicitly

1. In the root folder of your module, create a `META-INF` folder.
2. In the `META-INF` folder, create a manifest file `MANIFEST.MF` .
3. Add at least the following two headers to the `MANIFEST.MF` manifest file:
   - Bundle-ManifestVersion
   - Bundle-SymbolicName

   IntelliJ IDEA will now recognize the Osmorc facet for the module and will propose to add it. Confirm the addition.

Now you can start working on your OSGI application. Change your manifest files to export and import packages and Osmorc will create corresponding dependencies between the IntelliJ IDEA modules.

You may also want to take a look at the Settings-part of this documentation to learn how to define and use framework instances.

Set Up

This section describes how to set up a project with Osmorc.

In this part:

– How It Works
– Import Eclipse Workspace

How It Works

This topic describes how Osmorc works and what its limitations are.

OSGi and the IntelliJ IDEA Module System

OSGi consists of several layers. For an IDE to support development of OSGi applications, handling the module layer is the most important part because it has the biggest impact on how applications are developed.

The module layer defines a module system that allows for the specification of dependencies between the modules - called bundles - on a pretty fine grained level. It's possible to define which packages of a bundle are visible to other bundles and which other bundles or packages are needed by a bundle.

So if an IDE doesn't support OSGi's module layer, the application will not be compilable in that IDE until the developer recreates the bundle dependencies in the module system of the used IDE. While this is always possible to some extent, it is very inflexible and error prone.

Only the basic dependencies between bundles can be recreated with IntelliJ IDEA module system since it only allows to define which module uses which other modules. It is not possible to define which packages a module exports and it's also not possible to define which packages are needed by a module. Only dependencies on whole modules and not parts of modules can be expressed.

The Twofold Approach

Osmorc uses a twofold approach to the described problem which will work in most cases but will fail in some special cases.

Osmorc uses Eclipse Equinox to resolve bundle dependencies. For each bundle dependency a module dependency is created no matter which part of bundle B bundle A depends on, module A will have a module dependency on module B. So this first step doesn't care about which packages are exported or imported. So bundle A can see and use all packages and classes in bundle B. The fine grained rules are taken care of by the Invalid Import inspection. It checks whether the packages and classes from bundle B imported in a class of bundle A are actually exported by bundle B and imported by bundle A. If this check fails, the import or usage is marked as an error.

Limitations

The twofold approach implemented in Osmorc has several limitations. Inspection errors don't prevent compilation. So if the Invalid Import inspection marks usages and imports as errors, those errors won't prevent the compilation of the application. So it's possible to change a manifest file in a manner that makes the imports of a class invalid for Osmorc. If the class is not opened and checked by the developer and the Invalid Import inspection is also not via the Analyze | Inspect Code action, the developer will stay unaware of bugs introduced by his change. Only at runtime will the problem become apparent. So it's always a good idea to run the Invalid Import inspection on all classes of a project frequently.

Import Eclipse Workspace

Eclipse RCP and Eclipse Plugin projects are based on OSGi. At the core of an Eclipse application there is an OSGi runtime called Equinox. Osmorc's Equinox framework integrator provides support for working on applications using Equinox. At the time of this writing it is not possible to run and debug OSGi applications with Osmorc, but that is a planned feature for the near future. Currently Osmorc also doesn't contain any support for Eclipse specific features like extensions.

Currently it is possible to import an Eclipse workspace into IntelliJ IDEA and use Osmorc to resolve dependencies between the different OSGi bundles of that workspace. The outcome is that the created IntelliJ IDEA project can be compiled in IntelliJ IDEA without any errors with the possibility to edit the classes, manifest files, and other artifacts of the bundles.

From this topic you will learn how to import an Eclipse workspace into IntelliJ IDEA.

Tip Please, use the guide describing how to migrate Eclipse RCP applications to IntelliJ IDEA. This is a workaround that enables the developer of an Eclipse RCP application to work on that application with a pure IntelliJ IDEA without Osmorc. Projects changed as described in that guide won't be compatible with Osmorc. Osmorc adds OSGi support to IntelliJ IDEA making a great part of the workaround obsolete.

To import an Eclipse workspace into IntelliJ IDEA

1. Start importing an Eclipse workspace into IntelliJ IDEA as described in Importing Eclipse Project to IntelliJ IDEA .
2. Clear the Link created IntelliJ IDEA modules to Eclipse project files check box. Osmorc changes the dependencies between the modules created from the Eclipse projects in order to make them match the bundle dependencies expressed in the manifest files. Linking IntelliJ IDEA modules with Eclipse project files would result in unnecessary dependencies being introduced into the project files.
3. Remove the dependency to the global library ECLIPSE on any of the created modules, if one was added.

Osmorc only works on modules with its facet . Therefore after creating a project and modules for an Eclipse project, IntelliJ IDEA should detect the modules as being candidates for an Osmorc facet and display a message for each detection.

To add the Osmorc facet to all modules

1. Confirm adding Osmorc facet to each module.
2. To make sure that each module has an Osmorc facet, open the project structure , select the Modules node, and check that each module in the tree to the right has an Osmorc sub-node.
3. If you inadvertently deny the addition of the Osmorc facet, remove it or if it just does appear as a sub-node of the module node, add the Osmorc facet manually .

Now you have imported your Eclipse workspace into IntelliJ IDEA and you can start working on it in IntelliJ IDEA. One thing you haven't done yet, is transfer your target platform into IntelliJ IDEA. In Osmorc, the target platform is called a framework instance and you can define it in Osmorc's settings. Please read the Settings section to learn how to define different framework instances and how to choose one for a project.

Settings

There are two places where Osmorc's settings can be changed:

- Project specific and application wide settings are managed on the OSGi page of the Settings dialog box.
- Module specific settings are managed through the Osmorc module facet settings .

Project and Application Settings

The OSGi page

Facet Settings

In the facet settings you first have to decide whether you will edit the manifest files yourself or whether you want Osmorc to create it with bnd from the dependencies on other modules and libraries.

When you decide to maintain the manifest file manually, Osmorc creates dependencies on modules and the framework instance bundles in the module where the manifest file is changed. Here you take charge of the OSGi dependencies between your bundles and Osmorc tries to match them with corresponding module dependencies.

Letting Osmorc generate the manifest files for you is exactly the other way around. You develop your application as you would develop any other application in IntelliJ IDEA. You add dependencies on other modules and libraries and Osmorc tries to translate those dependencies into OSGi dependencies when running your bundles.

If you are maintaining the manifest file yourself, you have to tell Osmorc where it is to be found. You can change this location for each module or you can define a project default in the project settings and use it here.

Each time a module is compiled, Osmorc creates a bundle JAR for that module. This JAR is used when running an Osmorc run configuration. You can change how the JAR is named and where it is put, but for most cases the created default should be OK.

The created JAR contains the compilation output and the manifest file - either the manually maintained or the one created by Osmorc. Sometimes you will need to add some other files that need to be copied into the generated JAR. For Eclipse RCP bundles - or plugins as they are called there - you will need to add the `plugin.xml` . You can do this in the table listing additional JAR content. The first column takes the path to the source file relative to the content root of the module and the second column takes the path to the destination file. So since the plugin.xml of an Eclipse RCP plugin normally resides in the root of a plugin, you will type in `plugin.xml` for both columns of that entry. Future versions will provide a file chooser to choose the source file.

Note Currently the table for the additional JAR contents is very simple. It grows automatically as you type in new file definitions. There is always an empty row at the bottom that waits for new content. If you want to remove an entry, simply remove both the source and destination definition for it. It is then deleted automatically.

Project Settings

Project-specific Osmorc settings are managed in the Project Settings tab on the OSGi page of the Settings dialog box.

To configure the project-specific Osmorc settings

1. From the Framework Instance drop-down list, select the desired framework instance.
   The drop-down list contains all the framework instances defined for the currently running IntelliJ IDEA at the IDE level. If the framework instance you need is missing, switch to the Framework Definitions tab and define the required instance there.

   The drop-down list shows the names of the framework instances followed by the name of the used framework integrator in parentheses. In case you are opening a project created with another installation of IntelliJ IDEA which does not yet have the framework instance used by the project, `undefined` will be shown where normally the name of the framework integrator appears. If that is the case, switch to the Framework Definitions tab and define the framework instance.

2. To have the specified framework instance created for the module, select the Create and maintain the module Framework Instance check box.

3. Specify the default manifest file `MANIFEST.MF` location.

Framework Definitions

The OSGi specification has a number of implementations. Each implementation has a different set of base bundles, different ways to start them, and require different layout for the folder containing the binary bundles and their sources.

To cope with this diversity, Osmorc uses the notion of framework integrators and framework instances.

A framework integrator is used to integrate an instance of a specific framework implementation. Osmorc contains framework integrators for Eclipse Equinox , Knopflerfish , and Apache Felix . An extension point for framework integrators is also available. So integrators for framework implementations that are not directly supported by Osmorc can be developed by third parties.

A framework instance is a concrete installation of a framework implementation. As framework instances are normally not installed in the folder structure of a project, a framework instance with a specific name can be installed in different locations under different installations of IntelliJ IDEA. A project only knows the name of the framework instance it uses. Osmorc creates the connection to the framework instance if it exists as soon as the project is opened.

The framework definitions known by the currently used IntelliJ IDEA installation are listed on the OSGi page of the Settings dialog box.

Framework definitions are added, edited, and removed in the Framework Definitions tab of the OSGi page.

To define a new framework instance

1. Open the Settings dialog box , click the Osmorc node, and switch to the Framework Definitions tab.
2. Click the Add button.
3. In the Create New Framework Instance dialog box that opens select the type of framework integrator, specify a unique name and the base folder for the framework instance.

   Tip Please, refer to the descriptions of various framework integrators to find out what requirements this folder should meet.

Eclipse Equinox Framework Integrator

The Equinox framework integrator provides integration for Eclipse Equinox and its derivatives, such as Eclipse RCP DSK.

The base directory for Equinox framework instances is the `eclipse` folder of the installation or the folder containing the `plugins` folder, if the installation doesn't contain an `eclipse` folder.

Knopflerfish Framework Integrator

The Knopflerfish framework integrator provides integration for Knopflerfish.

The base directory for Knopflerfish framework instances is the installation folder of Knopflerfish. This installation folder should contain a `knopflerfish.org` folder, which in its turn should contain an `osgi` folder.

The integrator searches for bundles in the JARs in the child folder and for the matching sources in bundles.

Apache Felix Framework Integrator

The Felix framework integrator provides integration for Apache Felix.

As there is no single downloadable package containing both sources and binary bundles, you need to download all the items separately and specify their location.

To provide all the items the integrator expects to find, perform the following general steps

1. Download the binary ZIP or tar archive of Apache Felix and unpack its `felix-<version>` subfolder to the desired location.

   > **Note** At the time of this writing the current version of Apache Felix is 1.0.3, so the downloadable binary ZIP or tar contains a folder `felix-1.0.3`.

2. Specify the location of `felix-<version>` as the base folder for the integrator. The integrator loads the bundle JARs from the `bin` and `bundle` subfolders.

To enable access to the sources of the bundles in IntelliJ IDEA

1. In the base folder, create an `src` folder with the `bin` and `bundle` subfolders.

2. Copy all source ZIP archives for `felix.jar` into the `src/bin` folder. `main-1.0.3.zip` and `framework-1.0.3.zip` should be put here.

   > **Tip** Unfortunately, sources for some classes in `felix.jar` are missing in the archives and you'll have to additionally download some files, such as `org.osgi.core-1.0.0.zip`, and put them here to get the sources for all classes.

3. Store the source ZIP archives for the other bundles in the `src/bundle` folder.

   > **Warning!** Do not change the source ZIP of any of the bundles. The integrator expects to find the Felix specific folder structure inside them. A source ZIP named `mybundle-1.00.zip` will contain the sources in the `folder mybundle-1.00/src/main/java`.

Run Configurations

Osmorc provides the following run configuration types for running OSGi-based applications:

- OSGi Bundles

OSGi Bundles

The OSGi Bundles run configuration allows running bundles that don't have any dependencies on a specific OSGi framework implementation on any of the installed framework instances.

Currently there are limitations for this run configuration type:

- Only the core framework bundle will be loaded.
- Your bundles cannot depend on any other bundles of the framework.

To create an OSGi Bundles run configuration, perform the following general steps

- Open the Run/Debug Configurations dialog box, click the Add button +, and choose OSGi Bundles.
- In the dialog box that opens choose a framework instance to run your bundles on and specify the bundle to start with.

PHP-Specific Guidelines

PHP support includes:

- Possibility to create PHP files and classes from templates .
- Full PHP 5.6.x syntax support.
- Partial support for PHP 7 return type hints.
- Syntax highlighting .
- Error highlighting .
- Basic on-the-fly code completion .
- Resolution of `include` statements and file references, including references to `PHAR` archives.
  All the `PHAR` files from the current project and the specified include path are shown in the project tool window under the Project View/Libraries/PHAR node and available for browsing right there.
- Class Completion .
- Intention actions and quick fixes .
- Surrounding with code constructs `Ctrl+Alt+J` and `Ctrl+Alt+T` .
- Code inspections .
- Jump to declaration (`Ctrl+B`).
- Refactoring :
  - Rename (`Shift+F6`).
  - Move (`F6`).
  - Copy (`F5`).

PHP development support is provided through the PHP plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

This part describes some procedures that are specific for developing PHP applications and some preliminary steps that are required to configure PHP development environment.

To develop an application using PHP

Follow these general steps:

1. Configure the PHP development environment .
2. Enable PHP support in IntelliJ IDEA.
3. Start creating a project from scratch . On the first page of the New Project wizard , choose PHP in the left-hand pane, then choose PHP Empty Project in the right-hand pane.

   Tip To run and debug your application on a local Web server, create the project root below the Web server document root. Thus your application sources will be "visible" for the local Web server.
4. Create and configure the required data sources (see Managing Data Sources).
5. Populate the application using provided coding assistance.
6. Deploy the application.

   Tip With IntelliJ IDEA, you can flexibly configure deployment of PHP applications. For example, you can set up your PHP project on a local Web server from the very beginning, or develop and test an application locally and then upload it to a remote Web server, etc.
7. Run the application. You can do it in several ways:
   - From IntelliJ IDEA using a run configuration of the type PHP Web Application to view application output in a browser.
   - From IntelliJ IDEA using a PHP Console run configuration to view the application output in the Run tool window.
   - From IntelliJ IDEA, using a built-in Web server . This approach saves your time and effort because you do not need to deploy the application sources.

The following optional steps may be helpful:

- Set up unit testing in your project.
- Install and configure a debugging engine and specify the debugger options .
- Debug the application.
- Enable and use integration with PHP-specific command line tools .

Configuring PHP Development Environment

In this section:

- Configuring PHP Development Environment
  - Prerequisites
  - Configuring PHP development environment
- Installing an AMP Package
- Installing Components Separately

Prerequisites

Warning! This topic gives general guidelines in configuring environment for developing and testing PHP applications locally. These instructions by no means apply to configuring production environment, which is outside the scope of this Help.

PHP development requires the following software installed and configured:

- A Web server and a PHP engine are mandatory.
  Starting with version 5.4, PHP interpreters contain a built-in Web server. The server is by no means intended for production but for development and testing purposes only.
- A database server, if your application will use a database.
- A debugging tool, if you are going to debug your application.
- A command line tool, if you are going to run PHP commands from the command line.

Configuring PHP development environment

To set up the PHP development environment, follow these general steps:

1. Download, install, and configure the Web server, the PHP engine, and the MySQL server. Do one of the following:
   - Download, install, and configure the desired AMP package (Apache, MySQL, PHP).
   - Install and configure each component separately , then check your installation .

2. Optionally, perform these steps:
   - Install and configure a debugging engine .
   - Install or create and enable a PHP-specific command line tool .
   - Install and enable the PHPUnit tool .

Installing an AMP Package

AMP packages are operating system-specific. The most common ones are:

- XAMPP for Windows.

  Tip It is recommended that you use version 1.7.1 or later.

- The LAMP package compatible with the Linux distribution used.
- MAMP for OS X.

The installation procedure may differ depending on the operating system used, follow the installation instructions provided.

Warning! If you are using Windows Vista, avoid installing the package in the `Program Files` folder. This folder is write-protected by default, which means that no files can be placed on the server and further processed by the PHP engine.

To install and configure an AMP package

1. Download and install the desired AMP package.
2. Use the AMP control pane to start the components.
   If the server does not start, most likely a port conflict takes place. By default, the Apache HTTP server listens to port 80. This port can be already used by other services, for example, Skype. To solve the issue, update the server configuration file as follows:
   - Locate the line `Listen 80` and change it to, e.g., `Listen 8080` .
   - Locate the line `ServerName localhost:80` and change it accordingly, in this example to `ServerName localhost:8080` .

   Save the configuration file and restart the Web server.
3. To check your installation, open your browser and type the following URL address: `http://localhost:<port number>` . The AMP welcome page appears.

Installing Components Separately

The installation procedure for PHP environment components may differ depending on the operating system used, follow the installation instructions provided.

In this section:

- Installing and configuring the components separately
- Checking the Web server installation
- Checking the PHP engine installation

To install and configure each component separately

1. Download and install a Web server . The most common choice is the Apache HTTP server .
2. Download and install the PHP engine . During the installation, specify the Web server you are going to use and its home directory.

   Tip To enable the use of the MySQL database server, select the Complete installation option or select the MySQL and MySQLi items in the Extensions list on the corresponding page of the installation wizard..
3. Download, install, and configure a database server .

To check the Web server installation

- To make sure that the Web server has been installed correctly, start the server, open your browser, and type the following URL address: `http://localhost`
  The Test page for Apache installation should appear.

Tip If the server does not start, most likely a port conflict takes place. Resolve the conflict as during an AMP package installation.
- If you have changed the default port 80, specify the port number explicitly: `http://localhost:<port number>`

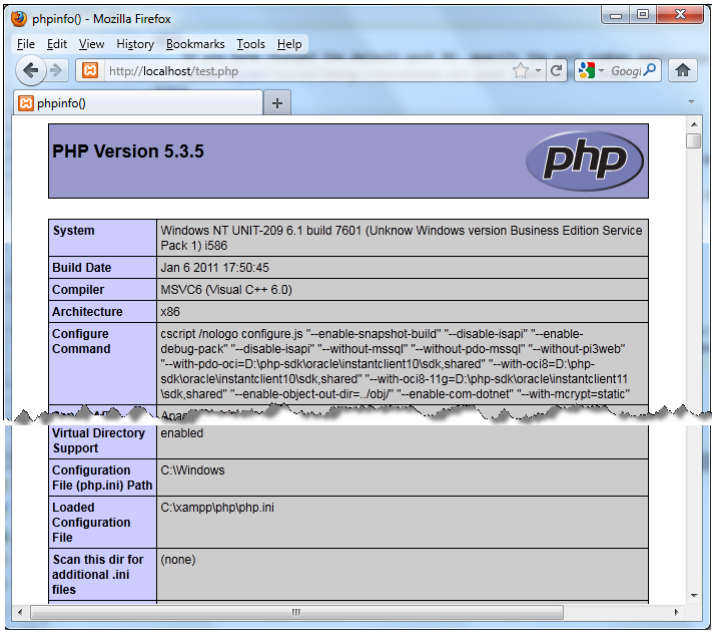
To check the PHP engine installation

Finally, make sure that the PHP engine has been installed successfully and interacts with the Web server correctly.

1. Using a text processor of your choice, create a file `test.php` and type the following text:

```
<?php
echo phpinfo();
?>
```

2. Save the file on the Web server, in the folder the PHP engine looks at.
3. Run the browser and enter the following URL address: `http://localhost:<port number>/test.php` . The page that opens should show detailed information on the version of PHP engine used, the location of the configuration files, etc.



Enabling PHP support

In this section:

- Enabling PHP Support
  - Introduction
  - Before you start
- Configuring Remote PHP Interpreters
- Configuring Local PHP Interpreters
- Enabling PHP Support in a Project

Introduction

After you have downloaded, installed, and configured the tools that make your development environment, you need to integrate them with IntelliJ IDEA and activate their support in the project you are going to work with. Basically, all you need to start is configure your PHP installation as an interpreter. You can have several PHP engines installed on your computer, or on remote hosts, or in Vagrant virtual development environments and configured as interpreters in IntelliJ IDEA, see Configuring Local PHP Interpreters and Configuring Remote PHP Interpreters . However, within a project, you can have only one of these interpreters activated.

Optionally, you can also enable support of other PHP development tools, see Configuring Synchronization with a Web Server , Configuring a Debugging Engine , PHP-Specific Command Line Tools , Testing PHP Applications .

Before you start

- Install and enable the PHP plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
- Synchronization with Web servers is provided via the Remote Hosts Access bundled plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Configuring Remote PHP Interpreters

On this page:

- Prerequisite
- Important notes
- Configuring a remote PHP interpreter on a host accessible through SSH connection
  - Before you start
  - Configuring a PHP Interpreter Using SSH Credentials
- Configuring a remote PHP interpreter in a Vagrant environment instance
  - Before you start
  - Configuring a PHP Interpreter in a Vagrant Instance
- Configuring a remote PHP interpreter on a remote host accessible through SFTP
  - Before you start
  - Configuring a remote PHP interpreter based on an SFTP server access configuration
- Configuring custom mappings

Prerequisite

Before you start working with remote interpreters, make sure that the SSH Remote Run plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Important notes

The term remote PHP interpreter denotes a PHP engine installed on a remote host or in a virtual environment set up in a Vagrant instance. The term remote PHP interpreter is used opposite to local PHP interpreters that are installed on your computer, see Configuring Local PHP Interpreters .

You can access a remote PHP interpreter in the following ways:

- Using SSH credentials to access the host where the PHP interpreter is installed.
- Through access to the corresponding Vagrant instance.
- According to a Server Access Configuration . This approach is also helpful if you are going to synchronize your project sources with the Web server on the target remote host.

Configuring a remote PHP interpreter on a host accessible through SSH connection

Before you start

1. Configure access to an `ssh` server on the target remote host and make sure this server is running.
2. Make sure the PHP Remote Interpreter repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Configuring a PHP Interpreter Using SSH Credentials

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click the … button next to the Interpreter drop-down list in the Development environment section.
3. In the Interpreters dialog box that opens, click the Add toolbar button + in the left-hand pane, then choose Remote on the context menu.
4. In the Configure Remote PHP Interpreter dialog box that opens, choose the SSH Credentials method.
5. Specify the name of the remote host and the port which the SSH server listens to. The default port number is 22.
6. Specify your credentials to access the remote host in accordance with the credentials received during the registration on the server. Type your user name and choose the authentication method:
   - To access the host through a password, choose Password from the Auth type drop-down list and type the password.
   - To access the host through a pair of SSH keys, choose Key pair, then specify the path to the file where your private key is stored or the passphrase if you have configured it during the generation of the key pair.

   To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . IntelliJ IDEA first attempts to retrieve path mappings itself by processing all the available application-level configurations. If IntelliJ IDEA finds the configurations with the same host as the one specified above, in the Host field, the mappings from these configurations are merged automatically. If no configurations with this host are found, IntelliJ IDEA displays an error message informing you that path mappings are note configured.

   To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the server access configuration in question, switch to the Mappings tab, and map local folders to folders on the server as described in Creating a Remote Server Configuration , section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.
7. Specify the location of the PHP executable file in accordance with the development environment configured on the selected remote host. By default IntelliJ IDEA suggests the `/usr/bin/php` folder. To specify another folder, click the Browse button … and choose the relevant folder in the dialog box that opens. Note that the PHP home directory must be open for edit.
8. When you click OK, IntelliJ IDEA checks whether the PHP executable is actually stored in the specified folder.
   - If no PHP executable is found, IntelliJ IDEA displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
   - If the PHP executable is found, you return to the Interpreters where the installation folder and the detected version of the PHP interpreter are displayed.
9. Optionally, customize the configuration settings of the installation in the Additional area. In the Debugger extension text box, specify the path to Xdebug. This enables IntelliJ IDEA to activate Xdebug when it is necessary is you have disabled it in the `php.ini` file, see Configuring Xdebug for Using in the On-Demand Mode .
   - In the Configuration options field, compose a string of configuration directives to be passed through the -d command line option and thus add new entries to the `php.ini` file. To do that, click the Browse button … next to the Configuration options field, and then create a list of entries in the Configuration Directives dialog box, that opens.
     - To add a new entry, click the Add button +. In the new line, that is added to the list, specify the name of the new entry and its value in the Name and Value text boxes respectively.

You can add as many entries as you need, just keep in mind that they will be transformed into a command line with its length limited to 256 characters.

- To delete an entry, select it in the list and click the Remove button.
- To change the order of entries, use the Up and Down buttons.

Upon clicking OK, you return to the Interpreters dialog box, where the entries are transformed into a command line.

Configuring a remote PHP interpreter in a Vagrant environment instance

Before you start

1. Make sure that Vagrant and Oracle's VirtualBox are downloaded, installed, and configured on your computer as described in Vagrant.
2. Make sure the Vagrant and PHP Remote Interpreter plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins. Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.
3. Make sure that the parent folders of the following executable files are added to the system PATH variable:
   - `vagrant.bat` or `vagrant` from your Vagrant installation. This should be done automatically by the Vagrant installer.
   - `VBoxManage.exe` or `VBoxManage` from your Oracle's VirtualBox installation.
4. Configure the PHP development environment in the Vagrant instance to be used.

Learn more about using Vagrant with IntelliJ IDEA in Vagrant.

Configuring a PHP Interpreter in a Vagrant Instance

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click the button next to the Interpreter drop-down list in the Development environment section.
3. In the Interpreters dialog box that opens, click the Add toolbar button in the left-hand pane, then choose Remote on the context menu.
4. In the Configure Remote PHP Interpreter dialog box that opens, choose the Vagrant method.
5. Specify the Vagrant instance folder which points at the environment you are going to use. Technically, it is the folder where the VagrantFile configuration file for the desired environment is located. Based on this setting, IntelliJ IDEA detects the Vagrant host and shows it as a link in the Vagrant Host URL read-only field.
   To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server. IntelliJ IDEA evaluates path mappings from the `VagrantFile` configuration file.
6. Specify the location of the PHP executable file in accordance with the development environment configured in the selected Vagrant instance. By default IntelliJ IDEA suggests the `/usr/bin/php` folder. To specify another folder, click the Browse button and choose the relevant folder in the dialog box that opens. Note that the PHP home directory must be open for edit.
7. When you click OK, IntelliJ IDEA checks whether the PHP executable is actually stored in the specified folder.
   - If no PHP executable is found, IntelliJ IDEA displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
   - If the PHP executable is found, you return to the Interpreters where the installation folder and the detected version of the PHP interpreter are displayed.
8. Optionally, customize the configuration settings of the installation in the Additional area. In the Debugger extension text box, specify the path to Xdebug. This enables IntelliJ IDEA to activate Xdebug when it is necessary is you have disabled it in the `php.ini` file, see Configuring Xdebug for Using in the On-Demand Mode.
   - In the Configuration options field, compose a string of configuration directives to be passed through the -d command line option and thus add new entries to the `php.ini` file. To do that, click the Browse button next to the Configuration options field, and then create a list of entries in the Configuration Directives dialog box, that opens.
     - To add a new entry, click the Add button. In the new line, that is added to the list, specify the name of the new entry and its value in the Name and Value text boxes respectively.
       You can add as many entries as you need, just keep in mind that they will be transformed into a command line with its length limited to 256 characters.
     - To delete an entry, select it in the list and click the Remove button.
     - To change the order of entries, use the Up and Down buttons.

     Upon clicking OK, you return to the Interpreters dialog box, where the entries are transformed into a command line.

Configuring a remote PHP interpreter on a remote host accessible through SFTP

Before you start

1. Make sure a `ssh` server is running on the target remote host and you have configured access to it.
2. Make sure the PHP Remote Interpreter repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins.
3. Make sure the Remote Hosts Access plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins.
4. Make sure you have at least one IntelliJ IDEA-wide server access configuration of the type SFTP to establish access to the target host. To make a configuration available in all IntelliJ IDEA projects, clear the Visible only for this project check box in the Deployment: Connection Tab. See Creating a Remote Server Configuration for details.

Configuring a remote PHP interpreter based on an SFTP server access configuration

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click the button next to the Interpreter drop-down list in the Development environment section.
3. In the Interpreters dialog box that opens, click the Add toolbar button in the left-hand pane, then choose Remote on the context menu.
4. In the Configure Remote PHP Interpreter dialog box that opens, choose the Deployment Configuration method.
5. From the Deployment Configuration drop-down list, choose the server access configuration of the type SFTP according to which you want IntelliJ IDEA to connect to the target host. If the settings specified in the chosen configuration ensure successful connection, IntelliJ IDEA displays the URL address of the target host as a link in the Deployment Host URL field.
   To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and

the URL addresses to access the copied data on the server . By default, IntelliJ IDEA retrieves path mappings from the chosen server access (deployment) configuration. If the configuration does not contain path mappings, IntelliJ IDEA displays the corresponding error message.

To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the relevant server access configuration, switch to the Mappings tab, and map the local folders to the folders on the server as described in Creating a Remote Server Configuration , section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.

6. Specify the location of the PHP executable file in accordance with the development environment configured on the selected remote host. By default IntelliJ IDEA suggests the `/usr/bin/php` folder. To specify another folder, click the Browse button and choose the relevant folder in the dialog box that opens. Note that the PHP home directory must be open for edit.
7. When you click OK, IntelliJ IDEA checks whether the PHP executable is actually stored in the specified folder.
   - If no PHP executable is found, IntelliJ IDEA displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
   - If the PHP executable is found, you return to the Interpreters where the installation folder and the detected version of the PHP interpreter are displayed.
8. Optionally, customize the configuration settings of the installation in the Additional area. In the Debugger extension text box, specify the path to Xdebug. This enables IntelliJ IDEA to activate Xdebug when it is necessary is you have disabled it in the `php.ini` file, see Configuring Xdebug for Using in the On-Demand Mode .
   - In the Configuration options field, compose a string of configuration directives to be passed through the -d command line option and thus add new entries to the `php.ini` file. To do that, click the Browse button next to the Configuration options field, and then create a list of entries in the Configuration Directives dialog box, that opens.
     - To add a new entry, click the Add button . In the new line, that is added to the list, specify the name of the new entry and its value in the Name and Value text boxes respectively.
       You can add as many entries as you need, just keep in mind that they will be transformed into a command line with its length limited to 256 characters.
     - To delete an entry, select it in the list and click the Remove button .
     - To change the order of entries, use the Up and Down buttons.

   Upon clicking OK, you return to the Interpreters dialog box, where the entries are transformed into a command line.
9. Click the Show phpinfo button to have IntelliJ IDEA display a separate information window where you can examine the installation details and view the list of loaded extension and configured options. Please note that the options specified in the Configuration Options field of the Interpreters dialog box are not listed.

Configuring custom mappings

If you use an interpreter accessible through SFTP connection or located on a Vagrant instance, the mappings are automatically retrieved from the corresponding deployment configuration or `Vagrantfile` . To specify additional mappings:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. From the Interpreter drop-down list, choose the remote interpreter for which you want to customize the mappings. The Path Mappings read-only field shows the path mappings retrieved from the corresponding deployment configuration or `Vagrantfile` , see the chapters above. To specify the custom mappings, click next to the Path Mappings field.
3. The Edit Project Path Mappings Dialog that opens, shows the path mappings retrieved from the deployment configuration or `Vagrantfile` . These mappings are read-only.
   - To add a custom mapping, click and specify the path in the project and the corresponding path on the remote runtime environment in the Local Path and Remote Path fields respectively. Type the paths manually or click and select the relevant files or folders in the dialog box that opens.
   - To remove a custom mapping, select it in the list and click .

Configuring Local PHP Interpreters

The term local PHP interpreter denotes a PHP engine installed on your computer opposite to remote PHP interpreters that can be installed on a remote host or in a virtual environment set up in a Vagrant instance, see Configuring Remote PHP Interpreters .

Configuring a local PHP interpreter in IntelliJ IDEA

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click the ... button next to the Interpreter drop-down list in the Development environment section.
3. In the Interpreters dialog box that opens, click the Add toolbar button + in the left-hand pane, then choose Local on the context menu. If you already have a local interpreter configured in IntelliJ IDEA, it is also shown on the menu and the menu item changes to Other Local.
4. In the right-hand pane of the dialog box, specify the PHP interpreter's settings.
   1. In the Name text box, type the identifier to distinguish the interpreter from others, for example, `php_installation_<version>` .
   2. Specify the PHP engine installation directory in the PHP Home field. Type the path manually or click the Browse button ... and choose the location in the Choose PHP Home dialog box, that opens.
      IntelliJ IDEA displays the version of the PHP engine detected in the specified folder and the debugger associated with this PHP engine in the `php.ini` file.
5. Optionally, customize the configuration settings of the installation in the Additional area. In the Debugger extension text box, specify the path to Xdebug. This enables IntelliJ IDEA to activate Xdebug when it is necessary is you have disabled it in the `php.ini` file, see Configuring Xdebug for Using in the On-Demand Mode .
   - In the Configuration options field, compose a string of configuration directives to be passed through the -d command line option and thus add new entries to the `php.ini` file. To do that, click the Browse button ... next to the Configuration options field, and then create a list of entries in the Configuration Directives dialog box, that opens.
     - To add a new entry, click the Add button +. In the new line, that is added to the list, specify the name of the new entry and its value in the Name and Value text boxes respectively.
       You can add as many entries as you need, just keep in mind that they will be transformed into a command line with its length limited to 256 characters.
     - To delete an entry, select it in the list and click the Remove button –.
     - To change the order of entries, use the Up and Down buttons.

     Upon clicking OK, you return to the Interpreters dialog box, where the entries are transformed into a command line.
6. Click the Show phpinfo button to have IntelliJ IDEA display a separate information window where you can examine the installation details and view the list of loaded extension and configured options. Please note that the options specified in the Configuration Options field of the Interpreters dialog box are not listed.

Enabling PHP Support in a Project

On this page:

- Creating an Empty PHP Project
- Creating a PHP Project from Existing Sources
- Choosing the PHP Interpreter and the Language Level Manually

Creating an Empty PHP Project

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. The Project Category and Options dialog opens.
2. In the left-hand pane, choose PHP, in the right-hand pane, choose PHP Empty Project, then click Next.
3. On the second page of the wizard:
   - In the Project Name text box,
   - In the Project Location text box, specify the path to the project folder where the project-related files will be stored.
   - In the PHP Language Level drop-down list, specify the PHP functionality scope to get coding assistance for. Each functionality scope is associated with PHP version that supports this functionality.
     No correlation between the PHP version used in the project and the language level is enforced. Although the language version of each interpreter is detected automatically, you can still tell IntelliJ IDEA to provide you with coding assistance that corresponds to another language level. However, if you attempt to use a code construct that is not supported by the specified language level, IntelliJ IDEA suggests a Switch to PHP <version> quick-fix .
   - From the Interpreter drop-down list, choose the PHP interpreter to use in the current project. The list contains all the currently configured local and remote PHP interpreters. See Configuring Local PHP Interpreters and Configuring Remote PHP Interpreters for details. You can also choose an interpreter or configure a new one later, see .
   - The Include Path area shows a list of paths to PHP-related items below the PHP home directory. The specified include paths will be used:
     - By the `require()` , `include()` , `fopen()` , `file()` , `readfile()` , and `file_get_contents()` functions when looking for files to use.
     - By IntelliJ IDEA when resolving references to included files.

     Use the Add + and Remove − buttons to manage the contents of the list. Use the Up and Down buttons to change the order of items in the list.

   When you click Finish, IntelliJ IDEA generates a project stub and opens it either in the current window or in a new one depending on your choice in the information dialog box tha opens.

See also Configuring PHP Namespaces in a Project

Creating a PHP Project from Existing Sources

Besides creating an empty PHP project, you can re-use your previously developed sources.

- When the files to re-use are already on your machine, all you need is just open the folder where they are stored:
  1. On the main menu, choose File | Open.
  2. In the dialog box that opens, select the folder where the required sources are stored.
- When the files to re-use are on a remote host:
  1. Configure synchronization with this remote host, see Configuring Synchronization with a Web Server .
  2. Download the sources to re-use, see Uploading and Downloading Files .
  3. Open the folder where the downloaded sources are stored and thus arrange them in a IntelliJ IDEA project. Alternatively, create an empty IntelliJ IDEA project and copy the downloaded source to it.

Choosing the PHP Interpreter and the Language Level Manually

To start PHP development in a project, you need to appoint one of the previously configured local or remote PHP interpreters to use and choose the PHP language level which determines the scope of PHP functionality to get coding assistance for. Currently PHP 5.3, PHP 5.4, PHP 5.5, and PHP 5.6 levels are supported.

To use PHP-related items from outside your project content root, configure include paths as described in Configuring Include Paths .

If you created your project through the New Project dialog, you have already specified the language level and possibly the interpreter and the include paths. However, if your project was created around some existing sources, you need to specify these settings manually.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks. Open the PHP page: choose File | Settings and in the Settings dialog box that opens, click PHP under Languages & Frameworks.
2. In the PHP page that opens, choose the PHP installation to use from the Interpreter drop-down list.
   - To make sure that the configuration you have chosen points at the relevant installation, click the Reload button next to the drop-down list. If no PHP engine is actually installed at the specified directory, IntelliJ IDEA displays the corresponding error message.
   - Click the Show phpinfo button to have IntelliJ IDEA display a separate information window where you can examine the installation details and view the list of loaded extension and configured options. Please note that the options specified in the Configuration Options field of the Interpreters dialog box are not listed.
3. In the PHP language level drop-down list, specify the PHP functionality scope to get coding assistance for. Each functionality scope is associated with PHP version that supports this functionality.
   No correlation between the PHP version used in the project and the language level is enforced. Although the language version of each interpreter is detected automatically, you can still tell IntelliJ IDEA to provide you with coding assistance that corresponds to another language level. However, if you attempt to use a code construct that is not supported by the specified language level, IntelliJ IDEA suggests a Switch to PHP <version> quick-fix .

Using Distributed Configuration Files (.htaccess)

On this page:

– Introduction
– Associating a name pattern with the distributed configuration file type
– Important notes

Introduction

IntelliJ IDEA provides syntax highlighting, formatting, code completion and documentation lookup while editing distributed configuration files .

Distributed configuration files are used to make directory-based changes to the HTTP Apache Server configuration and usually have the name `.htaccess`.

If you are using a file with the name `.htaccess`, IntelliJ IDEA recognizes it as distributed configuration file and provides full coding assistance for it, so no additional steps are required from your side.

If you want to use a file with another name, you need to associate this full name or the corresponding pattern with the distributed configuration file type . After that, IntelliJ IDEA will treat any file with the name matching the specified pattern as a distributed configuration file and process it accordingly.

Associating a name pattern with the distributed configuration file type

To associate a name pattern with the distributed configuration file type

1. Open the Settings/Preferences and click File Types.
2. In the File Types page that opens, select Apache config files from the Recognized File Types list.
3. In the Registered Patterns area, click +.
4. In the Add wildcard dialog box that opens, specify the pattern that defines the extensions of your distributed configuration files.
5. Click OK. IntelliJ IDEA returns you to the File Types page where the specified pattern is added to the Registered Patterns list.

Important notes

Please note the following:

– By default, the Registered Patterns list contains one item `htaccess`.
– To discard a pattern, select it in the list and click the Remove button.
– To change a pattern, select it in the list, click the Edit button, and update the pattern as necessary in the Add wildcard dialog box that opens.

Configuring Include Paths

Include paths are used for code completion and reference resolution in some functions/methods that use file paths as arguments (for example, `require()` or `include()`). You also need to configure PEAR as an include path to run the PHPUnit test framework from it when the Load from include path option is chosen on the PHPUnit page of the Settings / Preferences Dialog .

To configure include paths:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. In the PHP page that opens, click the Update include paths button.
3. In the Include paths area, that becomes available upon pressing the Update include paths button, configure the list of include paths.
   - To add a path to the list, click the Add button and select the required directory in the dialog box, that opens.
   - To remove a path from the list, select the required item and click the Remove button.

Configuring PHP Namespaces in a Project

On this page:

– Introduction

– Detecting namespace roots automatically

Introduction

IntelliJ IDEA comes with a configurable convention which specifies that the project root folder is also the root for the packages and namespaces. In other words: the project root folder is by default marked as Source and every directory created under it is considered a separate namespace. This complies with the PSR-0 , also known as the Autoloading Standard, which prescribes that classes and namespaces in PHP should match the directory and file structure, and vice-versa.

According to PSR-4 , any directory can be explicitly assigned a namespace prefix. With this project structure, autoloaders in different PHP frameworks become interoperable.

Detecting namespace roots automatically

When you open a project that contains at least one file with a namespace, IntelliJ IDEA displays a message with a proposition to set the namespace root.

Accordingly, when no namespace root has been configured yet and you create a class, IntelliJ IDEA proposes to configure the namespace root.

You can also trigger namespace root detection by choosing Code | Detect PSR-0 Namespace Roots on the main menu. The Directories dialog box that opens, shows the folders under the project root folder with the project root folder marked as Source, which means that it is the root for all the namespaces in it. Accept the settings by clicking OK or configure the namespace root manually as described below. Learn more about marking folders in Configuring Folders Within a Content Root .

Running PHP Applications

You can run your PHP application in several ways:

- From IntelliJ IDEA using a run configuration of the type PHP Web Application to view application output in a browser.
- From IntelliJ IDEA using a PHP Console run configuration to view the application output in the Run tool window.
- From IntelliJ IDEA, using a built-in Web server . This approach saves your time and effort because you do not need to deploy the application sources.

IntelliJ IDEA enables running entire PHP applications as well as particular classes or files .

To run an entire PHP application, perform these general steps

1. Define a PHP run configuration:
   - To launch the application via a console, create a PHP Console configuration.
   - To run the application on a local or a remote Web server, create a PHP Web Application configuration.
2. Deploy the PHP application on a local or remote server, if applicable.
3. Run the application with the required run configuration.

To run a particular PHP class or file, do one of the following

- Open the desired class or file in the editor and choose Run <file name> on the context menu or press `Ctrl+Shift+F10`.
- Select the class or file in the Project view and choose Run <file name> on the context menu of the selection.

PHP Built-In Web Server

On this page:

- Introduction
- Enabling a built-in Web server
- Creating a run configuration of the type PHP Built-in Server
- Running a PHP application on a built-in Web server

Introduction

Starting with version 5.4.0, PHP interpreters incorporate a Web server functionality. As soon as your project root folder or any other folder is declared the built-in Web server document root, all the files under this folder are submitted to the PHP interpreter for processing. All you need, is enable the built-in Web server in your project, create a run configuration of the type PHP Built-in Web server, and launch the application with this configuration.

Integration with the built-in Web server is supported in IntelliJ IDEA 11.1 and higher.

Enabling a built-in Web server

To enable a built-in Web Server

1. Download PHP 5.4.0 or higher.
2. Register the downloaded PHP interpreter in IntelliJ IDEA .
3. Open your project .
4. Appoint the PHP 5.4.0 interpreter configuration for use in the project.

Creating a run configuration of the type PHP Built-in Server

To create a built-in server run configuration

1. Open the Run/Debug Configuration dialog box by doing one of the following:
   - On the main menu, choose Run | Edit Configurations.
   - Press `Shift+Alt+F10`, then press `0` to display the Edit Configuration dialog box or select the configuration from the pop-up window and press `F4`.
2. Click + on the toolbar or press `Insert`. From the drop-down list, select the PHP Built-in Web Server configuration type. The PHP Built-in Web Server dialog box opens.
3. Specify the configuration name.
4. Specify the server connection settings:
   1. In the Host text box, type `localhost`.
   2. Use the Port spin box to specify the number of the port where the built-in web server runs. The port number should be set to 1024 or higher. By default this port is set to the default IntelliJ IDEA port through which IntelliJ IDEA accepts connections from services (`63342`).
5. Specify the folder that will be considered server document root . The folders that are under this document root in the project tree will be submitted to the PHP interpreter. The server document root folder will be accessed through HTTP at the above specified host:port.
   In the Document root text box, specify the path to the relevant folder. This can be the project root or any other folder under it.

   Optionally, customize the run configuration:
   - To have a specific script always executed upon the server start-up, select the Use router script check box and specify the path to the script in the text box.
   - To customize the behavior of the current PHP interpreter, in the Command Line area specify the options and arguments to be passed to the PHP executable file.

Running a PHP application on a built-in Web server

To run a PHP application on a built-in Web server

1. Run the application. To do that, select the built-in Web server run configuration from the Run/Debug Configuration drop-down list on the main tool bar, and then click the Run button ▶.
2. Launch the browser of your choice and open the starting page of your application by typing its URL address in the following format:

```
http://localhost:<port>/<path to the required file relative to the server document root>
```

Creating PHP Documentation Comments

On this page:

- Basics
  - Enabling generation of documentation comments
- Generating a PHPDoc block for a method or a function
- Creating tags in a PHPDoc comment block
- Inspecting PHPDoc comments
- Configuring formatting inside PHPDoc comments

Basics

IntelliJ IDEA creates stubs of PHPDoc blocks on typing the opening tag `/**` and pressing `Enter` or pressing `Alt+Insert` and appointing the method.function to document.

If this feature is applied to a method or a function, `@param`, `@access`, `@static`, `@throws`, `@return`, `@var`, and `@abstract` tags are created. In any other places IntelliJ IDEA adds an empty documentation stub.

If you need additional PHP-specific tags, IntelliJ IDEA provides code completion that suggests tag names that are relevant in the current context. If a certain tag has multiple values, the same code completion provides a list of available values.

In PHPDoc comments, IntelliJ IDEA supports formatting options in compliance with ZEND, PEAR, and other standards.

PHPDoc comments in your source code are available for the Quick Documentation Lookup feature and open for review on pressing `Ctrl+Q`.

To enable or disable generation of documentation comments

1. Open the Settings dialog box, expand the Editor node, then expand General node, and click the Smart Keys page.
2. In the Enter section, select or clear Insert documentation comment stub check box.

Generating a PHPDoc block for a method or a function

1. To invoke PHPDoc block generation, do one of the following:
   - Place the caret before the method or function declaration, type the opening block comment `/**`, and press `Enter`.
   - On the context menu anywhere in the editor, choose Generate, then choose Generate PHPDoc blocks, and choose the method or function to generate comments for.
   - Press `Alt+Insert`, then choose Generate PHPDoc blocks, and choose the method or function to generate comments for.

   IntelliJ IDEA analyzes the appointed method or function, where possible extracts the data for method parameters, return values, variables, etc., and on this basis generates a stub of a documentation block.
2. Describe the listed parameters and return values, where necessary.

> **Tip** IntelliJ IDEA checks syntax in the comments and treats it according to the PHP Inspections settings.

Creating tags in a PHPDoc comment block

IntelliJ IDEA analyzes the appointed method or function, where possible extracts the data for method parameters, return values, variables, etc., and on this basis generates a stub of a documentation block. If necessary, you can fill in the missing information.

1. In a PHPDoc block, select the desired empty line and press `Ctrl+Space`.
2. Select the relevant tag from the suggestion list.
3. If the entered tag has several values, press `Ctrl+Space` and select the desired value from the suggestion list.

Inspecting PHPDoc comments

IntelliJ IDEA provides a set of predefined code inspections targeted at PHPDoc blocks. These inspections check whether classes, methods, functions, variables, and constants are supplied with a PHPDoc comment and whether the tags in the comment match the documented item.

- To enable or disable an inspection:
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Inspections under Editor.
  2. On the Inspections page that opens, expand the PHPDoc node under the PHP node. The list of predefined inspections is displayed.
  3. To enable or disable an inspection, select or clear the check box next to it.

- To have IntelliJ IDEA check that PHPDoc comments are provided for all code constructs of a certain type:
  1. Select the check box next to the Missing PHPDoc Comment inspection.
  2. In the Options area, select the check boxes next to the required code construct type: class, method, function, variable, or constant.
  3. To suppress reporting a Missing PHPDoc Comment error if a method or function does not contain any parameters and/or return values, select the Skip if empty.

Configuring formatting inside PHPDoc comments

You can configure the appearance of PHPDoc comments and the presentation of class names in the Settings dialog box, on the PHPDoc tab of the Code Style. PHP under the Code Style node. Note that the tag for properties is no longer configurable, the default @var tag is inserted automatically. See https://github.com/phpDocumentor/fig-standards/pull/55 for details.

1. Open the Settings dialog box, click Code Style, then click PHP, and switch to the PHPDoc tab

1. Open the Settings dialog box, click Code Style, then click PHP, and switch to the PHP Doc tab.
2. Configure the alignment by selecting or clearing the check boxes in the tab.
3. Specify how you want IntelliJ IDEA to present class names for properties, function parameters, return and throws values, etc. by selecting or clearing the Use fully-qualified class names check box.

Viewing PHPDoc Comments

Quick Documentation Lookup helps you get quick information for any symbol, provided that this symbol has been supplied with Documentation comments in the applicable format. IntelliJ IDEA recognizes inline documentation created in accordance with the PHP Documentation format.

In this section:

– Viewing quick documentation .
– Changing font size in the quick documentation window .
– Toolbar of the quick documentation window .

To view documentation for a symbol at caret, do one of the following

– On the main menu, choose View | Quick Documentation Lookup.

– Press `Ctrl+Q`.

– Provided that the check box Show quick doc on mouse move in the editor settings is selected, just move your mouse pointer over the desired symbol.

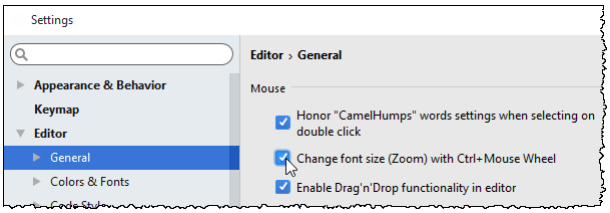
When you explicitly invoke code completion, then quick documentation for an entry selected in the suggestion list can be displayed automatically. The behavior of quick documentation lookup is configured in the Code Completion page of the Settings/Preferences dialog.

Tip Sequential pressing `Ctrl+Q` toggles the focus of the Quick Documentation pop-up window and Documentation Tool Window .

To change the font size of quick documentation, do one of the following

– Click in the upper-right corner of the quick documentation window, and move the slider.

– Rotate the mouse wheel while keeping the `Ctrl` key pressed.
Note that for this feature to work, you have to enable it in the General page of the editor settings.



Refer to the section Zooming in the Editor for details.

Quick Documentation Lookup Window

Toolbar of the quick documentation lookup

The Quick Documentation Lookup window helps navigate to the related symbols via hyperlinks, and provides a toolbar for moving back and forth through the already navigated pages, change font size, and viewing documentation in an external browser.

When pinned, the Quick Documentation Lookup turns into Documentation Tool Window , with the corresponding sidebar icon, and more controls.

Icon	Shortcut	Description
	`Left` or `Right`	Switch to the previous or next documentation page (e.g. after using hyperlinks). **Note** On an OS X computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.
	`Shift+F1`	View external documentation in the default browser.
	`F4`	Switch to the item (e.g. source) that corresponds to the documentation page currently shown.
		Turn the Auto-update from source option on or off. When the option is on, the information in the tool window is synchronized with your navigation in the editor and other places in the UI.
		Click this icon to show font size slider. Move the slider to increase or decrease the font size in the quick documentation window as required.

Using PHP Code Sniffer Tool

On this page:

- Introduction
- Before you start
- Choosing the Code Sniffer script to use
- Configuring a local Code Sniffer script
- Configuring a Code Sniffer associated with a PHP interpreter
- Configuring advanced PHP Code Sniffer options
- Configuring PHP Code Sniffer as a IntelliJ IDEA inspection
- Choosing a custom coding style to check your code against
- Sharing a custom coding style with the team
- Running Code Sniffer in the batch mode
- Excluding files from Code Sniffer inspection

Introduction

In addition to built-in coding assistance, IntelliJ IDEA provides code style check through integration with the PHP Code Sniffer tool, which validates your code for consistence with a coding standard of your choice. You can appoint one of the predefined coding standards or use your own previously defined coding standard with the root directory outside the default PHP Code Sniffer's Standards directory. Moreover, you can share your custom coding style with your team fellows.

To use PHP Code Sniffer right from IntelliJ IDEA instead of from a command line, you need to register it in IntelliJ IDEA and configure it as a IntelliJ IDEA code inspection . Once installed and enabled in IntelliJ IDEA, the tool is available in any opened PHP file, so no steps are required from your side to launch it. The on-the-fly code check is activated upon every update in the file thus making it extremely easy to get rid of problems reported by PHP Code Sniffer.

Errors and warnings reported by PHP Code Sniffer on-the-fly are displayed as pop-up messages. When the tool is run in the batch mode, the errors and warnings are displayed in the Inspection tool window. Anyway, each message has the `phpcs` prefix to distinguish it from IntelliJ IDEA internal inspections.

Before you start

Make sure the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Choosing the Code Sniffer script to use

You can use PHP Code Sniffer local scripts or scripts associated with PHP interpreters. There can be a number of local and remote PHP interpreters, the one specified on the PHP page of the Settings dialog box is considered Project Default. Learn more about configuring PHP interpreters in Configuring Remote PHP Interpreters or in Configuring Local PHP Interpreters .

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Code Sniffer under PHP.
2. From the Configuration drop-down list in the Development Environment area, choose the script to use:
   - To use the script associated with a specific remote PHP interpreter, choose the name of this interpreter.
   - To use the script associated with the default project interpreter, that is, the one chosen on the PHP page of the Settings dialog box, choose By default project interpreter.
   - To use a local script, choose Local. In this case the local Code Sniffer will be executed no matter which PHP interpreter - local or remote - is used in the project. Note that there can be only one Local configuration for Code Sniffer because IntelliJ IDEA runs a script (`phpcs.bat` for Windows or `phpcs` for Linux) which contains a path to a PHP engine.

Configuring a local Code Sniffer script

1. Make sure the PEAR package manager is installed on your machine.
2. Download and install the PHP Code Sniffer . To check it, switch to the directory with the `pear.bat` file and run the following command: `phpcs --version`
   If the tool is available, you will get a message in the following format: `PHP_CodeSniffer version <version> (stable) by Squiz Pty Ltd. (http://www.squiz.net)`

   To have code checked against your own custom coding standard, create it . Store the rules and the `ruleset.xml` file that points to them in the coding standard root directory.
3. Register the local PHP Code Sniffer script in IntelliJ IDEA:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Code Sniffer under PHP.
   2. On the Code Sniffer page that opens, click ... next to the Configuration drop-down list.
   3. In the Code Sniffer dialog box that opens, specify the location of the Code Sniffer executable `phpcs.bat` or `phpcs` in the PHP Code Sniffer path text box. Type the path manually or click the Browse button ... and select the path in the dialog box, that opens.
      To check that the specified path to `phpcs.bat` or `phpcs` ensures interaction between IntelliJ IDEA and Code Sniffer, that is, the tool can be launched from IntelliJ IDEA and IntelliJ IDEA will receive problem reports from it, click the Validate button. This validation is equal to running the `phpcs --version` command. If validation passes successfully, IntelliJ IDEA displays the information on the detected Code Sniffer version.

Configuring a Code Sniffer associated with a PHP interpreter

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Code Sniffer under PHP.
2. On the Code Sniffer page that opens, click ... next to the Configuration drop-down list. The Code Sniffer dialog box opens showing a list of all the configured Code Sniffer scripts in the left-hand pane, one of them is of the type Local and others are named after the PHP interpreters with which the scripts are associated. Click + on the toolbar.
3. In the Code Sniffer by Remote Interpreter dialog box that opens, choose the remote PHP interpreter to use the associated script from. If the list does not contain a relevant interpreter, click ... and configure a remote interpreter in the Interpreters dialog box as described in Configuring Remote PHP Interpreters .
   When you click OK, IntelliJ IDEA brings you back to the Code Sniffer dialog box where the new Code Sniffer configuration is added to the list and the right-hand pane

shows the chosen remote PHP interpreter, the path to the Code Sniffer associated with it, and the advanced PHP Code Sniffer options.

Configuring advanced PHP Code Sniffer options

IntelliJ IDEA provides the ability to specify advanced PHP Code Sniffer options and thus fine tune the PHP Code Sniffer process behavior depending on the configuration of your computer and the rule sets used.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Code Sniffer under PHP.
2. In the Maximum number of messages per file text box, set the upper limit for the total number of messages to be reported for a file. All the messages above this limit will be rejected and IntelliJ IDEA will display the following warning right in the code:

   ```
   Too many PHP Code Sniffer messages
   ```

3. In the Tool process timeout text box, specify how long you want IntelliJ IDEA to wait for a result from PHP Code Sniffer, whereupon the process is killed to prevent excessive CPU and memory usage.

Configuring PHP Code Sniffer as a IntelliJ IDEA inspection

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Inspections under Editor.
2. On the Inspections page that opens, select the PHP Code Sniffer validation check box under the PHP node.
3. On the right-hand pane of the page, configure the PHP Code Sniffer tool using the controls in the Options area:
   1. From the Severity drop-down list, choose the severity degree for the Code Sniffer inspection. The selected value determines how serious the detected discrepancies will be treated by IntelliJ IDEA and presented in the inspection results .
   2. In the Coding standard drop-down list, appoint the coding style to check your code against. The list contains all the coding standards installed inside the main `PHP_CodeSniffer` directory structure.
      Use one of the predefined coding standards or choose Custom to appoint your own standard .
   3. Optionally, select the Ignore warnings check box to have only errors reported and suppress reporting warnings. This option is equal to the `-n` command line argument.

Choosing a custom coding style to check your code against

You can have code checked against your own previously defined coding standard with the root directory outside the default PHP Code Sniffer's Standards directory. This root directory should contain the rules themselves and the `ruleset.xml` file that points to them.

1. Open the Settings dialog box , and click Inspections. The Inspections page opens. Select the PHP Code Sniffer validation check box under the PHP node.
2. From the Coding standard drop-down list, choose Custom.
3. Click the Browse button ....
4. In the Custom Coding Standard dialog box that opens, specify the path to the root directory of your own coding standard in the Root directory. Type the path manually or click the Browse button ... and choose the relevant folder in the dialog that opens .
   The selected root directory should contain the `ruleset.xml` file that points to the rules.

Sharing a custom coding style with the team

1. Put the root directory of your coding standard under the project root .
2. Configure Code Sniffer as a IntelliJ IDEA inspection .
3. Appoint your coding standard .
4. At the top of the Inspections page, select the Share Profile check box.
5. On the Version Control page of the Settings dialog box, put the root directory of your coding standard under version control .

Running Code Sniffer in the batch mode

1. To run the inspection , choose Code | Inspect code on the main menu. Specify the inspection scope and profile.
2. View the inspection results in the Inspection tool window. Errors and warnings reported by PHP Code Sniffer have the `phpcs` prefix to distinguish them from IntelliJ IDEA internal inspections.

Excluding files from Code Sniffer inspection

When waiting for Code Sniffer response exceeds the limit specified in the Tool process timeout field on Code Sniffer page, IntelliJ IDEA suggests adding the file to the ignore list. This list is shown on the Code Sniffer page in the Ignored files area. For each file, IntelliJ IDEA displays its name and location.

- To delete a file from the list and have Code Sniffer process it again, select the file and click the Remove file button .
- To remove all the files from the list, click the Clean the list button .

Using PHP Mess Detector

On this page:

- Introduction
- Before you start
- Choosing the Mess Detector script to use
- Configuring a local Mess Detector script
- Configuring a Mess Detector associated with a PHP interpreter
- Specifying advanced PHP Mess Detector options
- Configuring PHP Mess Detector as a IntelliJ IDEA inspection
- Specifying the rules to apply
- Running Mess Detector in the batch mode
- Excluding files from Mess Detector inspection

Introduction

In addition to built-in coding assistance, IntelliJ IDEA provides checking the source code through integration with the PHP Mess Detector tool, which detects potential problems related to code size, inconsistency, unused code, violation of naming conventions, poor design, etc.

You can have predefined rules applied or define your own custom set of rules .

Before you start

Make sure the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Choosing the Mess Detector script to use

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Mess Detector under PHP.
2. From the Configuration drop-down list in the Development Environment area, choose the script to use:
   - To use the script associated with a specific remote PHP interpreter, choose the name of this interpreter.
   - To use the script associated with the default project interpreter, that is, the one chosen on the PHP page of the Settings dialog box, choose By default project interpreter.
   - To use a local script, choose Local. In this case the local Mess Detector will be executed no matter which PHP interpreter - local or remote - is used in the project. Note that there can be only one Local configuration for Mess Detector because IntelliJ IDEA runs a script (`phpmd.bat` for Windows or `phpmd` for Linux) which contains a path to a PHP engine.

Configuring a local Mess Detector script

1. Download PHP Mess Detector. The easiest ways to do that is use the Composer Dependency Manager, see Using Composer Dependency Manager .
2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Mess Detector under PHP.
3. On the Mess Detector page that opens, click … next to the Configuration drop-down list.
4. In the Mess Detector dialog box that opens, specify the location of the Mess Detector executable `phpmd.bat` or `phpmd` in the PHP Mess Detector path text box. Type the path manually or click the Browse button … and select the path in the dialog box, that opens. If you installed the tool through Composer, the default location is `<project root folder>\vendor\bin\phpmd.bat` .
   To check that the specified path to `phpmd.bat` or `phpmd` ensures interaction between IntelliJ IDEA and Mess Detector, that is, the tool can be launched from IntelliJ IDEA and IntelliJ IDEA will receive problem reports from it, click the Validate button. This validation is equal to running the `phpmd --version` command. If validation passes successfully, IntelliJ IDEA displays the information on the detected Mess Detector version.

Configuring a Mess Detector associated with a PHP interpreter

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Mess Detector under PHP.
2. On the Mess Detector page that opens, click … next to the Configuration drop-down list. The Mess Detector dialog box opens showing a list of all the configured Mess Detector scripts in the left-hand pane, one of them is of the type Local and others are named after the PHP interpreters with which the scripts are associated. Click + on the toolbar.
3. In the Mess Detector by Remote Interpreter dialog box that opens, choose the remote PHP interpreter to use the associated script from. If the list does not contain a relevant interpreter, click … and configure a remote interpreter in the Interpreters dialog box as described in Configuring Remote PHP Interpreters .
   When you click OK, IntelliJ IDEA brings you back to the Mess Detector dialog box where the new Mess Detector configuration is added to the list and the right-hand pane shows the chosen remote PHP interpreter, the path to the Mess Detector associated with it, and the advanced PHP Mess Detector options.

Specifying advanced PHP Mess Detector options

IntelliJ IDEA provides the ability to specify advanced PHP Mess Detector options and thus fine tune the PHP Mess Detector process behavior depending on the configuration of your computer and the rule sets used.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Mess Detector under PHP. The Mess Detector page opens.
2. In the Maximum number of messages per file text box, set the upper limit for the total number of messages to be reported for a file. All the messages above this limit will be rejected and IntelliJ IDEA will display the following warning right in the code:

```
Too many PHP Mess Detector messages
```

3. In the Tool process timeout text box, specify how long you want IntelliJ IDEA to wait for a result from PHP Mess Detector, whereupon the process is killed to prevent

excessive CPU and memory usage.

Configuring PHP Mess Detector as a IntelliJ IDEA inspection

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Inspections under Editor.
2. On the Inspections page that opens, select the PHP Mess Detector validation check box under the PHP node.
3. On the right-hand pane of the page, configure the PHP Mess Detector tool using the controls in the Options area:
   1. From the Severity drop-down list, choose the severity degree for the Mess Detector inspection. The selected value determines how serious the detected discrepancies will be treated by IntelliJ IDEA and presented in the inspection results .
   2. Appoint the rules to apply .

Specifying the rules to apply

You can have predefined rules applied or define your own custom set of rules.

1. Open the Settings dialog box , and click Inspections.
2. On the Inspections page that opens, select the PHP Mess Detector validation check box under the PHP node.
3. Do one of the following:
   - To use a predefined rules, in the Options area, select the check boxes next to the validations to be performed. For more details on predefined rules, see http://phpmd.org/rules/index.html .
   - To use a custom ruleset:
     1. Create and save one or several ruleset files. A valid ruleset file is an `.xml` file that contains the root element `<ruleset>` with the attribute `name` . For more details on custom rulesets, see http://phpmd.org/documentation/creating-a-ruleset.html .
     2. In the Custom rulesets area, click the Add Rule button + and select the relevant rule definition file in the dialog box that opens. When you click OK, a new item is added to the Custom rulesets list, where the File field shows the location of the selected ruleset file and the Name field shows the ruleset name retrieved from the attribute `name` within the `<ruleset>` tag.

Running Mess Detector in the batch mode

1. To run the inspection , choose Code | Inspect code on the main menu. Specify the inspection scope and profile.
2. View the inspection results in the Inspection tool window. Errors and warnings reported by PHP Mess Detector have the `phpmd` prefix to distinguish them from IntelliJ IDEA internal inspections.

Excluding files from Mess Detector inspection

When waiting for Mess Detector response exceeds the limit specified in the Tool process timeout field on Mess Detector page, IntelliJ IDEA suggests adding the file to the ignore list. This list is shown on the Mess Detector page in the Ignored files area. For each file, IntelliJ IDEA displays its name and location.

- To delete a file from the list and have Mess Detector process it again, select the file and click the Remove file button .
- To remove all the files from the list, click the Clean the list button .

Deploying PHP Applications

On this page:

- Introduction
- Configuring deployment
- Uploading files automatically

Introduction

Synchronization with Web servers is provided via the Remote Hosts Access bundled plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

With IntelliJ IDEA, you can flexibly configure deployment of PHP applications. For example, you can set up your PHP project on a local Web server from the very beginning, or develop and test an application locally and then upload it to a remote Web server, etc.

By default, the sources are deployed according to the chosen configuration when you run your application. However, you can have IntelliJ IDEA upload files automatically every time they are changed. In this case the default server configuration is used.

Configuring deployment

To configure deployment

1. Create a local or remote Web server configuration .
2. Define mappings between local folders, remote folders, and URL addresses.
3. Customize the upload procedure by specifying additional options.

Uploading files automatically

To have updated files uploaded automatically, do one of the following

- Choose Tools | Deployment | Automatic Upload on the main menu.
- Open the Settings dialog box , and click Options below the Deployment node. In the Options dialog box, select the Upload changed files automatically to the default server check box.

Debugging PHP Applications

In this part:

- Configuring a Debugging Engine
- PHP Debugging Session
- Multiuser Debugging via Xdebug Proxies

Configuring a Debugging Engine

This section describes how to configure two most popular tools used in PHP debugging:

– Xdebug

– Zend Debugger

These tools cannot be used simultaneously because they block each other. To avoid this problem, you need to update the corresponding sections in the `php.ini` file. To open the active `php.ini` file in the editor:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click … next to the Interpreter field.
3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.

In this part:

– Configuring Xdebug

– Configuring Zend Debugger

– Validating the Configuration of a Debugging Engine

Configuring Xdebug

Configuring the Xdebug tool involves:

- Downloading and installing the Xdebug debugging engine
- Enabling Xdebug integration with the PHP engine
- Integrating Xdebug with IntelliJ IDEA
- Configuring Xdebug for Using in the On-Demand Mode

Downloading and installing the Xdebug debugging engine

Download the Xdebug extension compatible with your version of PHP and save it in the `php/` folder.

- The location of the `php/` folder is defined during the installation of the PHP engine .
- If you are using an AMP package , the Xdebug extension may be already installed. Follow the instructions in the `xdebug.txt` .

Enabling Xdebug integration with the PHP engine

1. Open the active `php.ini` file in the editor:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. On the PHP page that opens, click … next to the Interpreter field.
   3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.
2. To enable Xdebug, locate the `[Xdebug]` section in the `php.ini` file and update it as follows:

```
[Xdebug]
zend_extension="<path to php_xdebug.dll>"
xdebug.remote_enable=1
xdebug.remote_port="<the port for Xdebug to listen to>" (the default port is 9000)
xdebug.profiler_enable=1
xdebug.profiler_output_dir="<AMP home\tmp>"
```

3. To enable multiuser debugging via Xdebug proxies , locate the `xdebug.idekey` setting and assign it a value of your choice. This value will be used to register your IDE on Xdebug proxy servers.
4. Save and close the `php.ini` file.

Integrating Xdebug with IntelliJ IDEA

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. Check the Xdebug installation associated with the selected PHP interpreter:
   1. On the PHP page, choose the relevant PHP installation from the Interpreter drop-down list and click the Browse button … next to the field. The list shows all the PHP installations available in IntelliJ IDEA, see Enabling PHP Support .
   2. The Interpreters dialog box that opens shows the following:
      - The version of the selected PHP installation.
      - The name and version of the debugging engine associated with the selected PHP installation (Xdebug or Zend Debugger). If no debugger is configured, IntelliJ IDEA shows `Debugger: Not installed` .

   Alternatively, open the Xdebug checker , paste the output of the `phpinfo()` , and click Analyze my phpinfo() output.

   Learn more about checking the Xdebug installation in Validating the Configuration of a Debugging Engine .
3. Define the Xdebug behaviour. Click Debug under the PHP node. On the Debug page that opens, specify the following settings in the Xdebug area:
   - In the Debug Port text box, appoint the port through which the tool will communicate with IntelliJ IDEA. This must be exactly the same port number as specified in the `php.ini` file:

```
xdebug.remote_port = <port_number>
```

     By default, Xdebug listens on port 9000.
   - To have IntelliJ IDEA accept any incoming connections from Xdebug engines through the port specified in the Debug port text box, select the Can accept external connections check box.
   - Select the Force break at the first line when no path mapping is specified check box to have the debugger stop as soon as it reaches and opens a file that is not mapped to any file in the project on the Servers page. The debugger stops at the first line of this file and Debug Tool Window. Variables shows the following error message: Cannot find a local copy of the file on server <path to the file on the server> and a link Click to set up mappings. Click the link to open the Resolve Path Mappings Problem dialog box and map the problem file to its local copy.
     When this check box cleared, the debugger does not stop upon reaching and opening an unmapped file, the file is just processed, and no error messages are displayed.
   - Select the Force break at the first line when the script is outside the project check box to have the debugger stop at the first line as soon as it reaches and opens a file outside the current project. With this check box cleared, the debugger continues upon opening a file outside the current project.
4. In the External Connections area, specify how you want IntelliJ IDEA to treat connections received from hosts and through ports that are not registered as server configurations .
   - Ignore external connections through unregistered server configurations: Select this check box to have IntelliJ IDEA ignore connections received from hosts and through ports that are not registered as server configurations . When this check box is selected, IntelliJ IDEA does not attempt to create a server configuration automatically.

- Detect path mappings from deployment configurations:
  - When this check box is selected, IntelliJ IDEA attempts to retrieve path mappings for debugging in a remote environment from the server access configuration (deployment configuration).
  - When the check box is cleared, you have to specify the path mappings manually.

  See Validating the Configuration of a Debugging Engine , Web Server Debug Validation Dialog , and Configuring Remote PHP Interpreters for details.
- Break at first line in PHP scripts: Select this check box to have the debugger stop as soon as connection between it and IntelliJ IDEA is established (instead of running automatically until the first breakpoint is reached). Alternatively turn on the Run | Break at first line in PHP scripts option on the main menu.
- Max. simultaneous connections: Use this spin box to limit the number of external connections that can be processed simultaneously.

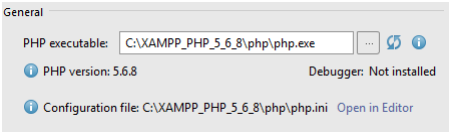
Configuring Xdebug for Using in the On-Demand Mode

Starting with version 2016.2, IntelliJ IDEA supports the On-Demand mode where you can disable Xdebug for your global PHP installation, and have it enabled automatically on demand only when you are debugging your command-line scripts or when you need code coverage reports. This lets your command line scripts (including Composer and unit tests) run much faster.

1. Disable Xdebug for command-line scripts:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. From the PHP executable drop-down list, choose the relevant PHP interpreter and click … next to it. In the Interpreters dialog box that opens, click the Open in Editor link next to the Configuration file: <path to php.ini> file. Close all the dialog boxes and switch to the tab where the `php.ini` file is opened.
   3. In the `php.ini` file, find the `[XDebug]` section and comment the following line in it by adding `;` in preposition:

      ```
      ;[XDebug]
                          ;zend_extension = "C:\XAMPP_PHP_5_6_8\php\ext\php_xdebug-2.3.2-5.6-vc11.dll"
      ```

   4. Open the Interpreters dialog box and click next to the PHP executable field. IntelliJ IDEA informs you that debugger is not installed:

2. To enable IntelliJ IDEA to activate Xdebug when it is necessary, specify the path to it in the Debugger extension text box, in the Additional area. Type the path manually or click … and select the location in the dialog box that opens.

Configuring Zend Debugger

Configuring the Zend Debugger tool involves:

- Downloading and installing Zend Debugger
- Enabling Zend Debugger integration with the PHP engine
- Integrating Zend Debugger with IntelliJ IDEA
- Configuring Zend Debugger for Using in the On-Demand Mode

Downloading and installing Zend Debugger

1. Download the Zend Debugger package which corresponds to your operating system.
2. Locate the `ZendDebugger.so` (Unix) or `ZendDebugger.dll` (Windows) file in the directory which corresponds to your version of PHP (e.g. 4.3.x, 4.4.x, 5.0.x, 5.1.x, 5.2.x, 5.3.x, 5.4.x).
3. Copy the file to your Web server in a location that is accessible by the Web server.

Enabling Zend Debugger integration with the PHP engine

1. Open the active `php.ini` file in the editor:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. On the PHP page that opens, click ... next to the Interpreter field.
   3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.
2. Locate or create the `[Zend]` section.
3. To load the Zend Debugger extension, add one of the following lines inside the `[Zend]` section depending on your operating system:
   - Linux and OS X:

     ```
     zend_extension=<full_path_to_ZendDebugger.so>
     ```

   - Windows:

     ```
     zend_extension_ts=<full_path_to_ZendDebugger.dll>
     ```

   - Windows non-thread safe:

     ```
     zend_extension=<full_path_to_ZendDebugger.dll>
     ```

   Warning! The Windows non-thread safe binary file is only used with Zend Core 2.0.

4. To enable access to Zend Debugger from PhpStorm, add the following lines:

   ```
   zend_extension=<full_path_to_zend_debugger_extension>
   zend_debugger.allow_hosts=127.0.0.1
   zend_debugger.expose_remotely=allowed_host
   zend_debugger.tunnel_min_port=<any integer value above 1024>
   zend_debugger.tunnel_max_port=<any integer value below 65535>
   ```

   The value of the `zend_debugger.allow_hosts` parameter is the IPs of your machine to connect to the server debugger. It could be a comma-separated list of IPs in the format `X.X.X.X` (for example, 192.168.0.6).

   Tip For a thread-safe Windows binary use the `zend_extension_ts parameter` instead of `zend_extension`.

5. Restart your Web server.
6. To check that the Zend Debugger has been installed and configured correctly, create a file with the following contents:

   ```
   <?php
   phpinfo();
   ?>
   ```

   Open the page that corresponds to the file in the browser. The output should contain a Zend Debugger section.

Integrating Zend Debugger with IntelliJ IDEA

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. Check the Zend Debugger installation associated with the selected PHP interpreter:

1. On the PHP page, choose the relevant PHP installation from the Interpreter drop-down list and click the Browse button … next to the field. The list shows all the PHP installations available in IntelliJ IDEA, see Enabling PHP Support .
2. The Interpreters dialog box that opens shows the following:
   - The version of the selected PHP installation.
   - The name and version of the debugging engine associated with the selected PHP installation (Xdebug or Zend Debugger). If no debugger is configured, IntelliJ IDEA shows `Debugger: Not installed` .

   Learn more about checking the Zend Debugger installation in Validating the Configuration of a Debugging Engine .
3. Define the Zend Debugger behaviour. Click Debug under the PHP node. On the Debug page that opens, specify the following settings in the Zend Debugger area:
   - In the Debug Port text box, appoint the port for IntelliJ IDEA to communicate with the tool through. Type the port number within the tunnel specified in the `php.ini` file through `zend_debugger.tunnel_min_port` and `zend_debugger.tunnel_max_port` . For details, see http://files.zend.com/help/previous-version/Zend-Server-4-Community-Edition/zenddebugger.html
   - To have IntelliJ IDEA accept any incoming connections from Zend Debugger engines through the port specified in the Debug port text box, select the Can accept external connections check box.
   - To use a debugger toolbar in the browser, specify the port through which the debugger settings are passed to the browser in the Settings broadcasting port text box.
4. In the External Connections area, specify how you want IntelliJ IDEA to treat connections received from hosts and through ports that are not registered as server configurations .
   - Ignore external connections through unregistered server configurations: Select this check box to have IntelliJ IDEA ignore connections received from hosts and through ports that are not registered as server configurations . When this check box is selected, IntelliJ IDEA does not attempt to create a server configuration automatically.
   - Detect path mappings from deployment configurations:
     - When this check box is selected, IntelliJ IDEA attempts to retrieve path mappings for debugging in a remote environment from the server access configuration (deployment configuration).
     - When the check box is cleared, you have to specify the path mappings manually.

     See Validating the Configuration of a Debugging Engine , Web Server Debug Validation Dialog , and Configuring Remote PHP Interpreters for details.
   - Break at first line in PHP scripts: Select this check box to have the debugger stop as soon as connection between it and IntelliJ IDEA is established (instead of running automatically until the first breakpoint is reached). Alternatively turn on the Run | Break at first line in PHP scripts option on the main menu.
   - Max. simultaneous connections: Use this spin box to limit the number of external connections that can be processed simultaneously.
5. To block requests from the Z-Ray system if they annoy you by invoking the IntelliJ IDEA debugger too often, select the Ignore Z-Ray system requests check box.

Configuring Zend Debugger for Using in the On-Demand Mode

Starting with version 2016.2, IntelliJ IDEA supports the On-Demand mode where you can disable Zend Debugger for your global PHP installation, and have it enabled automatically on demand only when you are debugging your command-line scripts or when you need code coverage reports. This lets your command line scripts (including Composer and unit tests) run much faster.

1. Disable Zend Debugger for command-line scripts:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. From the PHP executable drop-down list, choose the relevant PHP interpreter and click … next to it. In the Interpreters dialog box that opens, click the Open in Editor link next to the Configuration file: <path to php.ini> file. Close all the dialog boxes and switch to the tab where the `php.ini` file is opened.
   3. In the `php.ini` file, find the `[Zend]` section and comment the following lines in it by adding `;` in preposition: Linux and OS X:

      ```
      zend_extension=<full_path_to_ZendDebugger.so>
      ```

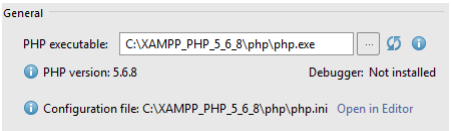
      Windows:

      ```
      zend_extension_ts=<full_path_to_ZendDebugger.dll>
      ```

      Windows non-thread safe:

      ```
      zend_extension=<full_path_to_ZendDebugger.dll>
      ```

      Warning! The Windows non-thread safe binary file is only used with Zend Core 2.0.
   4. Open the Interpreters dialog box and click next to the PHP executable field. IntelliJ IDEA informs you that debugger is not installed:

2. To enable IntelliJ IDEA to activate Zend Debugger when it is necessary, specify the path to it in the Debugger extension text box, in the Additional area. Type the path manually or click … and select the location in the dialog box that opens.

Validating the Configuration of a Debugging Engine

IntelliJ IDEA can validate your configuration of Xdebug or Zend Debugger and tell you if some setting is missing (for example, `xdebug.remote_enable`) or inconsistent with other settings. When configuring the PHP interpreter for a project, IntelliJ IDEA informs you whether a debugger is installed in your local PHP development environment and reports on the Xdebug or Zend Debugger version used. For details, see Configuring a Debugging Engine and Enabling PHP Support.

You can also get more detailed information about the debugging engine on a local or remote Web server.

1. On the main menu, choose Run | Web Server Debug Validation.
2. In the Web Server Debug Validation Dialog that opens, choose the Web server to validate the debugger on.
   - Choose Local Web Server or Shared Folder to check a debugger associated with a local Web server.
     - Path to Create Validation Script: In this field, specify the absolute path to the folder under the server document root where the validation script will be created. For Web servers of the type Inplace, the folder is under the project root.
       The folder must be accessible through http.
     - URL to Validation Script: In this text box, type the URL address of the folder where the validation script will be created. If the project root is mapped to a folder accessible through `http`, you can specify the project root or any other folder under it.
   - Choose Remote Web Server to check a debugger associated with a remote server.
     - Path to Create Validation Script: In this field, specify the absolute path to the folder under the server document root where the validation script will be created. The folder must be accessible through http.
     - Deployment Server: In this field, specify the server access configuration of the type Local Server or Remote Server to access the target environment. For details see Configuring Synchronization with a Web Server.
       Choose a configuration from the drop-down list or click the Browse button … in the Deployment dialog.
3. Click Validate to have IntelliJ IDEA create a validation script, deploy it to the target remote environment, and run it there.

PHP Debugging Session

Before you start debugging, make sure that you have a debugging engine installed and configured properly. IntelliJ IDEA supports debugging with two most popular tools: Xdebug and Zend Debugger. These tools cannot be used simultaneously because they block each other. To avoid this problem, you need to update the corresponding sections in the `php.ini` file as described in Configuring Xdebug and Configuring Zend Debugger.

To open the active `php.ini` file in the editor:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click ... next to the Interpreter field.
3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.

For more information on configuring debugging engines, see Xdebug Installation Guide and Zend Debugger Installation Guide.

With IntelliJ IDEA, a PHP debugging session can be initiated either through a debug configuration or without it. The latter approach is also called Zero-configuration debugging. IntelliJ IDEA supports three main ways to initiate a PHP debugging session:

- You create a PHP Web Application debug configuration, and then IntelliJ IDEA uses its settings to launch the application, open the browser, and activate the debugging engine.
- You create a PHP HTTP Request debug configuration, IntelliJ IDEA generates a request on its base, and then accesses a specific page through this request.
- Zero-configuration debugging, when no debug configuration is created at all. Instead, you open the starting page of your PHP application in the browser manually and then activate the debugging engine from the browser, while IntelliJ IDEA listens to incoming debugger connections.

No matter which method you choose, you can specify the scripts requests to which you want IntelliJ IDEA to ignore during debugging. This approach can be useful, when your application contains scripts that use AJAX. Suppose you have a `menu-ajax-script.php` that "reloads" a part of your web page. This script works properly so you do not need to debug it. However, this script is still requested during the debugging session. To have incoming connections to this script ignored, add the `menu-ajax-script.php` script to the skipped paths list. You can also group such scripts into folders and add these folders to the "ignore list".

When using Xdebug, you can also debug PHP applications in the multiuser mode via Xdebug proxy servers.

In this part:

- Creating a PHP Debug Server Configuration
- Debugging with a PHP Web Application Debug Configuration
- Zero-Configuration Debugging
- Debugging a PHP HTTP Request

Creating a PHP Debug Server Configuration

The term PHP debug server configuration denotes the settings that ensure HTTP access for debugging engines to interact with local and remote Web servers and set correspondence between files on the server and their local copies in the IntelliJ IDEA project. The settings from debug server configurations are used when debugging with run/debug configurations of the type PHP Web Application or PHP Remote Debug and during Zero-Configuration Debugging sessions.

You can create a debug server configuration manually from scratch or import some settings from a server access (deployment) configuration (see Configuring Synchronization with a Web Server for details).

On this page:

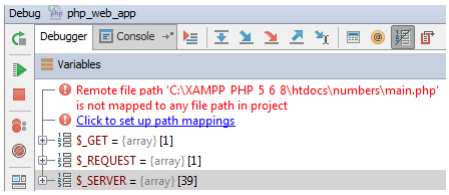
– Defining a Debug Server Configuration Manually
– Importing Settings from a Server Access (Deployment) Configuration

Defining a Debug Server Configuration Manually

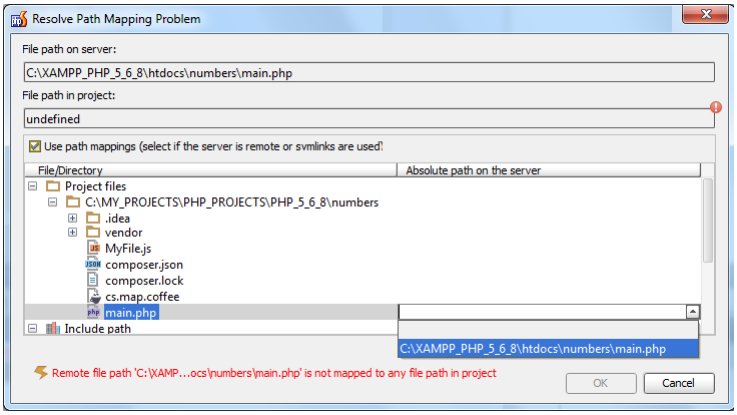
1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Servers under PHP.
   Alternatively, click … next to the Server drop-down list in the Run/Debug Configuration: PHP Web Application dialog box.

   In either case, the Servers page opens with the left-hand pane showing all the available debug server configurations. Click + on the toolbar and specify the debug server settings in the right-hand pane.
2. Specify the debug server configuration name.
3. Specify the host where the application is run and the port to access it.
4. From the Debugger drop-down list, choose the debugging engine to use.
5. Specify how IntelliJ IDEA will set up correspondence between files on the server and their local copies. Based on these mappings, IntelliJ IDEA will open local copies of currently processed files.
   Path mappings in PHP Debug Server configurations look very similar to the path mappings in server access (deployment configurations. Unfortunately, they cannot be reused, as deployment configurations uses relative paths while PHP Debug Servers configurations rely on absolute paths.
   – Select the Use path mappings check box if you are working on a remote Web server, that is, when the Web server is on a physically remote host, or the Web server is installed on your machine but your project is outside the Web server document root. Also select the check box if you are using symlinks.
     Map the absolute paths to the files and folders on the server with absolute paths to your project files in the local file system using the Path on server and File/Directory fields respectively.
     – File / Directory: This read-only field displays the files and folders of the current project. Select a file or a folder to be used as the local copy.
     – Path on server: In this filed, specify the absolute path to the file or folder on the target server to which the selected local file or folder corresponds. Type the path manually or select it from the drop-down list.
   – Clear the Use path mappings check box if you are working right on your Web server so your project root is under the server document root. In this case the absolute paths to the files on the Web server and the absolute paths to the corresponding files in your project are the same.

   If you do not specify any path mappings and start debugging an application that is not under the server document root, IntelliJ IDEA displays an error message:



   The Click to set up path mappings link brings up the Resolve Path Mappings Problem dialog box, where you can define the path mappings:



   When you click OK and leave the dialog box, IntelliJ IDEA selects the Use path mappings check box on the Servers page automatically.
6. Select the Shared check box to share the debug server configuration across a team. The host/port settings and the path mappings are stored in the `.idea/php.xml` file is available to all team members through a version control system. Note that mappings are shared only for directories inside the project.

Importing Settings from a Server Access (Deployment) Configuration

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Servers under PHP.
   Alternatively, click … next to the Server drop-down list in the Run/Debug Configuration: PHP Web Application dialog box.

   In either case, the Servers page opens with the left-hand pane showing all the available debug server configurations. Click on the toolbar. The Import from Deployment Configuration Dialog dialog box opens.

2. From the Deployment drop-down list, choose the server access configuration (deployment configuration) to copy the server access settings from. The list contains all the available deployment configurations. To create a new configuration, click … and specify new settings in the Deployment: Connection Tab dialog box that opens.
3. Depending on the type of the server access configuration chosen from the Deployment drop-down list, specify one of the following:
   - For an FTP, SFTP, or FTPS server access configuration, specify the absolute path to the server deployment root. This path will be added as a prefix to the path from the Root Path text box on the Deployment: Connection Tab .
     If you are not sure about this absolute path, you can open the Remote Host tool window, choose the required deployment configuration, position the cursor at the root folder, and choose Copy Path on the context menu, see Accessing Files on Web Servers for details. Alternatively, contact your hosting provider.
   - For Local or mounted folder, type the absolute path to the server root as specified in the Folder field of the Creating a PHP Debug Server Configuration dialog box.
   - For Inplace Server configurations no mappings are required because the local and remote paths are the same in this case.
4. The Preview area shows the host/port and the path mappings retrieved from the chosen server access configuration (deployment configuration).
   When you choose the deployment configuration to use, the Absolute path on the server field shows relative paths mapped to the project files and folders in the chosen configuration, that is, paths to files and folders relative to the deployment root. As you specify the absolute path to the deployment root (the server root for FTP/SFTP/FTPS or the mounted folder), the contents of the field are updated automatically and finally the field shows absolute paths on the server.

Debugging with a PHP Web Application Debug Configuration

On this page:

- Introduction
- Preparing the Debugging Engine
- Setting breakpoints
- Creating a PHP Web Application debug configuration
- Initiating a debugging session and examining the suspended program
- Specifying scripts to skip requests to

Introduction

In this debugging mode, IntelliJ IDEA fully controls the debugging process: it launches the application, opens the browser, and activates the debugging engine according to a PHP Web Application debug configuration.

A PHP Web Application debug configuration tells IntelliJ IDEA the URL address to access the starting page of the application, the browser to open the starting page in, and the debug server configuration to use.

You can also specify the scripts requests to which you want IntelliJ IDEA to ignore during debugging. This approach can be useful, when your application contains scripts that use AJAX. Suppose you have a `menu-ajax-script.php` that "reloads" a part of your web page. This script works properly so you do not need to debug it. However, this script is still requested during the debugging session. To have incoming connections to this script ignored, add the `menu-ajax-script.php` script to the skipped paths list. You can also group such scripts into folders and add these folders to the "ignore list".

Preparing the Debugging Engine

Before you start debugging, make sure that you have a debugging engine installed and configured properly. IntelliJ IDEA supports debugging with two most popular tools: Xdebug and Zend Debugger. These tools cannot be used simultaneously because they block each other. To avoid this problem, you need to update the corresponding sections in the `php.ini` file as described in Configuring Xdebug and Configuring Zend Debugger.

To open the active `php.ini` file in the editor:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click … next to the Interpreter field.
3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.

For more information on configuring debugging engines, see Xdebug Installation Guide and Zend Debugger Installation Guide.

Setting breakpoints

Breakpoints are source code markers used to trigger actions during a debugging session. Typically, the purpose behind setting a breakpoint is to suspend program execution to allow you to examine program data. However, IntelliJ IDEA can use breakpoints as triggers for a variety of different actions. Breakpoints can be set at any time during the debugging process. Your breakpoints don't affect your source files directly, but the breakpoints and their settings are saved with your IntelliJ IDEA project so you can reuse them across debugging sessions.

1. Place the caret on the desired line of the source code.
   Breakpoints can be set in the PHP context inside `*.php`, `*.html`, and files of other types. Only executable lines are valid locations for line breakpoints. Comments, declarations, and empty lines are not valid locations for the
2. Do one of the following:
   - Click the left gutter area at a line where you want to toggle a breakpoint.
   - On the main menu, choose Run | Toggle Line Breakpoint.
   - Press `Ctrl+F8`.

Creating a PHP Web Application debug configuration

1. Open the Run/Debug Configuration dialog box by doing one of the following:
   - On the main menu, choose Run | Edit Configurations.
   - Press `Shift+Alt+F10`, then press `0` to display the Edit Configuration dialog box or select the configuration from the pop-up window and press `F4`.
2. Click + on the toolbar or press `Insert`. From the drop-down list, select the PHP Web Application configuration type. The PHP Web Application dialog box opens.
3. Specify the configuration name.
4. Choose the applicable debug server configuration from the Server drop-down list or click the Browse button … and define a debug server configuration in the Servers dialog box that opens as described in Creating a PHP Debug Server Configuration
5. In the Start URL text box, type the server path to the file that implements the application starting page. Specify the path relative to the server configuration root (The server configuration root is the highest folder in the file tree on the local or remote server accessible through the server configuration. For in-place servers, it is the project root.). The read-only field below shows the URL address of the application starting page. The URL address is composed dynamically as you type.
6. Specify the browser to open the application in. Choose a configured browser from the Browser drop-down list or click the Browse button … and specify another browser in the Web Browsers dialog box that opens.

Initiating a debugging session and examining the suspended program

1. To start debugging, click the Debug button on the toolbar.
2. As soon as the debugger suspends on reaching the first breakpoint, examine the application by analyzing frames. A frame corresponds to an active method or function call and stores the local variables of the called method or function, the arguments to it, and the code context that enables expression evaluation. All currently active frames are displayed on the Frames pane of the Debug tool window . where you can switch between them and analyze the information stored therein in the Variables and Watches panes. For more details, see the section Examining Suspended Program .
3. Continue running the program and examine its frames as soon as it is suspended again.
   - To control the program execution manually, step through the code using the commands under the Run menu or toolbar buttons: Step Into (`F7`), Step Out (`Shift+F8`), Step Over (`F8`), and others. For more details, see Stepping Through the Program .

- To have the program run automatically up to the next breakpoint, resume the session by choosing Run | Resume Program or pressing `F9`

Specifying scripts to skip requests to

This approach can be useful, when your application contains scripts that use AJAX. Suppose you have a `menu-ajax-script.php` that "reloads" a part of your web page. This script works properly so you do not need to debug it. However, this script is still requested during the debugging session. To have incoming connections to this script ignored, add the `menu-ajax-script.php` script to the skipped paths list. You can also group such scripts into folders and add these folders to the "ignore list".

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the PHP node under Languages & Frameworks, and then click Skipped Paths under Debug.
2. On the Skipped Paths page that opens, configure an "ignore list" of scripts and folders with scripts not to be invoked if IntelliJ IDEA receives incoming connections to them.
   - To add a new entry to the list, click the Add button + or press `Alt+Insert`. Then click the Browse button and in the dialog box that opens choose the file or folder to skip connections to.
   - To remove an entry from the list, select it and click the Remove button – or press `Alt+Delete`. The script will be now executed upon receiving requests to it.
3. To have IntelliJ IDEA inform you every time it receives a request to a script to be skipped, select the Notify about skipped paths check box.

Zero-Configuration Debugging

In case of zero-configuration debugging, you do not need to create any debug configuration. Instead, you open the starting page of your PHP application in the browser manually, and then activate the debugging engine from the browser, while IntelliJ IDEA listens to incoming debugger connections.

To enable starting and stopping the debugging engine from the browser manually, you need to set a special `GET` / `POST` or `COOKIE` parameter. You can do it manually in the `php.ini` configuration file or generate the Start Debugger/ Stop Debugger bookmarklets on the toolbar of your browser. These bookmarklets provide control over the debugger cookie, through them you will activate and deactivate the debugger.

For more details about setting the parameters manually, see Starting the Debugger for Xdebug and Zend Debugger GET Request Parameters for Zend Debugger.

You can also specify the scripts requests to which you want IntelliJ IDEA to ignore during debugging. This approach can be useful, when your application contains scripts that use AJAX. Suppose you have a `menu-ajax-script.php` that "reloads" a part of your web page. This script works properly so you do not need to debug it. However, this script is still requested during the debugging session. To have incoming connections to this script ignored, add the `menu-ajax-script.php` script to the skipped paths list. You can also group such scripts into folders and add these folders to the "ignore list".

On this page:

- Preparing the Debugging Engine
- Setting the Breakpoints
- Generating the Start Debugger/Stop Debugger Bookmarklets
- Initiating a Debugging Session and Examining the Suspended Program
- Troubleshooting
  - No Path Mappings are Configured
  - No Debug Server is Configured
  - A Script Is not Suspended
- Specifying the Scripts to Skip Requests to
- Starting a Debugging Session from the Command Line
  - Starting a Script with Xdebug
  - Starting a Script with Zend Debugger
  - Configuring Path Mappings
  - Troubleshooting

Preparing the Debugging Engine

Before you start debugging, make sure that you have a debugging engine installed and configured properly. IntelliJ IDEA supports debugging with two most popular tools: Xdebug and Zend Debugger. These tools cannot be used simultaneously because they block each other. To avoid this problem, you need to update the corresponding sections in the `php.ini` file as described in Configuring Xdebug and Configuring Zend Debugger.

To open the active `php.ini` file in the editor:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click ... next to the Interpreter field.
3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.

For more information on configuring debugging engines, see Xdebug Installation Guide and Zend Debugger Installation Guide.

Setting the Breakpoints

Breakpoints are source code markers used to trigger actions during a debugging session. Typically, the purpose behind setting a breakpoint is to suspend program execution to allow you to examine program data. However, IntelliJ IDEA can use breakpoints as triggers for a variety of different actions. Breakpoints can be set at any time during the debugging process. Your breakpoints don't affect your source files directly, but the breakpoints and their settings are saved with your IntelliJ IDEA project so you can reuse them across debugging sessions.

1. Place the caret on the desired line of the source code.
   Breakpoints can be set in the PHP context inside `*.php`, `*.html`, and files of other types. Only executable lines are valid locations for line breakpoints. Comments, declarations, and empty lines are not valid locations for the
2. Do one of the following:
   - Click the left gutter area at a line where you want to toggle a breakpoint.
   - On the main menu, choose Run | Toggle Line Breakpoint.
   - Press `Ctrl+F8`.

Generating the Start Debugger/Stop Debugger Bookmarklets

These bookmarklets will appear on the toolbar of your browser. They provide control over the debugger cookie, through them you will activate and deactivate the debugger.

1. Enable the Bookmarks toolbar in your browser by doing one of the following depending on the browser type:
   - In Firefox, choose View | Toolbar | Bookmarks Toolbar.
   - In Chrome, choose Bookmarks | Show bookmarks bar.
2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Debug under PHP.
3. On the Debug page, that opens, click the Use debugger bookmarklets to initiate debugger from your favorite browser link.
4. On the Zend Debugger & Xdebug bookmarklets page that opens, check the debugging engine settings and click Generate. The bookmarks for listed debugging-related actions are generated.
5. Drag the generated links to the bookmark toolbar in your browser.

Initiating a Debugging Session and Examining the Suspended Program

1. Generate the bookmarklets to toggle the debugger through. These bookmarklets will appear on the toolbar of your browser. They provide control over the debugger cookie, through them you will activate and deactivate the debugger.
   1. Enable the Bookmarks toolbar in your browser by doing one of the following depending on the browser type:
      - In Firefox, choose View | Toolbar | Bookmarks Toolbar.
      - In Chrome, choose Bookmarks | Show bookmarks bar.
   2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Debug under PHP.
   3. On the Debug page, that opens, click the Use debugger bookmarklets to initiate debugger from your favorite browser link.
   4. On the Zend Debugger & Xdebug bookmarklets page that opens, check the debugging engine settings and click Generate. The bookmarks for listed debugging-related actions are generated.
   5. Drag the generated links to the bookmark toolbar in your browser.
2. Set the breakpoints , where necessary.
3. Toggle the Start Listen PHP Debug Connections button so it changes to . After that IntelliJ IDEA starts listening to the port of the debugging engine used in the current project . Ports for debuggers are set at the IntelliJ IDEA level in the Debug dialog box (File | Settings | PHP | Debug).
4. Open the starting page of your application in the browser, choose the Start debugger bookmark to activate the debugging engine from the browser, re-load the current page (the starting page of the application), and then return to IntelliJ IDEA.
5. As soon as the debugger suspends on reaching the first breakpoint, examine the application by analyzing frames. A frame corresponds to an active method or function call and stores the local variables of the called method or function, the arguments to it, and the code context that enables expression evaluation. All currently active frames are displayed on the Frames pane of the Debug tool window . where you can switch between them and analyze the information stored therein in the Variables and Watches panes. For more details, see the section Examining Suspended Program .
6. Continue running the program and examine its frames as soon as it is suspended again.
   - To control the program execution manually, step through the code using the commands under the Run menu or toolbar buttons: Step Into (F7), Step Out (Shift+F8), Step Over (F8), and others. For more details, see Stepping Through the Program .
   - To have the program run automatically up to the next breakpoint, resume the session by choosing Run | Resume Program or pressing F9

Troubleshooting

During a debugging session you may encounter one of the following problems:

No Path Mappings are Configured

In some cases you may get one of the following error messages: Remote file path path/to/script/on/the/server.php is not mapped to any file path in project or The script path/to/script/on/the/server.php is outside the project. This means that IntelliJ IDEA is not sure which local file corresponds to the specified remote file path because you have not specified the path mappings in the Debug server configuration.

Even if path mappings in a Debug Server configuration look very similar to path mappings in a Deployment configuration, they cannot be reused, because Deployment configurations require paths to be relative, and Debug Server configurations rely on absolute paths.

Click the link and configure path mappings in the dialog box that opens, as described in Creating a PHP Debug Server Configuration .

No Debug Server is Configured

If you initiate a debugging session without having configured a Debug server, upon establishing connection IntelliJ IDEA displays the Incoming Connection Dialog where suggests importing mappings from a server access configuration (deployment configuration). If you choose Import mappings from deployment, IntelliJ IDEA tries to detect the most suitable deployment configuration, preselects it in the Deployment drop-down list, and the Preview area shows the absolute path to the project file which corresponds to the currently executed script according to the mappings from the selected configuration.

If IntelliJ IDEA does not detect a relevant configuration:

1. Choose the most suitable configuration from the drop-down list or click and create a new configuration in the Deployment dialog box that opens, whereupon the new configuration is added to the list.
2. In the Deployment root text box, type the absolute path to the server root folder

You can also select the Manually choose local file or project option, in this case IntelliJ IDEA displays the project tree view where you can select a project file and map the currently executed script to it. You can also select and map the entire project.

To continue debugging with the imported or manually specified configuration settings, click Accept.

A Script Is not Suspended

Establishing a Zero-Configuration debugging session may fail, with no breakpoints hit and therefore the script not suspended. This may happen if if the path mappings are not configured or configured erroneously, or if you have not set any breakpoints. In the latter case, enabling selecting the Break at First Line in PHP Scripts check box in the External Connections area or turning the Run | Break at First Line in PHP Scripts option on the main menu may also help.

To have IntelliJ IDEA display a notification if the script is not suspended, select the Notify if debug session was finished without being stopped check box in the Advanced Settings area on the Debug page of the Settings dialog box.

Specifying the Scripts to Skip Requests to

This approach can be useful, when your application contains scripts that use AJAX. Suppose you have a `menu-ajax-script.php` that "reloads" a part of your web page. This script works properly so you do not need to debug it. However, this script is still requested during the debugging session. To have incoming connections to this script ignored, add the `menu-ajax-script.php` script to the skipped paths list. You can also group such scripts into folders and add these folders to the "ignore list".

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the PHP node under Languages & Frameworks, and then click Skipped Paths under Debug.
2. On the Skipped Paths page that opens, configure an "ignore list" of scripts and folders with scripts not to be invoked if IntelliJ IDEA receives incoming connections to them.
   - To add a new entry to the list, click the Add button or press Alt+Insert . Then click the Browse button and in the dialog box that opens choose the file or folder to skip connections to.

- To remove an entry from the list, select it and click the Remove button or press `Alt+Delete`. The script will be now executed upon receiving requests to it.

3. To have IntelliJ IDEA inform you every time it receives a request to a script to be skipped, select the Notify about skipped paths check box.

Starting a Debugging Session from the Command Line

You can start debugging a PHP CLI script from the command line, having IntelliJ IDEA listen for incoming debugger connections.

1. Set the breakpoints where necessary.
2. On the toolbar, toggle the Start Listening for PHP Debug Connections button or choose Run | Start Listening for PHP Debug Connections on the main menu.
3. Start the script with debugger options depending on the debugging engine you are using - Xdebug or Zend Debugger.

Starting a Script with Xdebug

Xdebug has various configuration options which can be used to let the PHP interpreter reach out to IntelliJ IDEA. These parameters have to be passed to the PHP interpreter using the `-d` command line switch. A more convenient is to set an environment variable you do not need to provide the `-d` switches all the time.

Do one of the following:

- Launch PHP with several switches:

```
php -dxdebug.remote_enable=1 -dxdebug.remote_mode=req -dxdebug.remote_port=9000 -dxdebug.remote_host=127.0.0.1 -dxdebug.remote_connect_back=0 p
```

- Set an environment variable that configures Xdebug:
  - `set XDEBUG_CONFIG="remote_enable=1 remote_mode=req remote_port=9000 remote_host=127.0.0.1 remote_connect_back=0"` for Windows
  - `export XDEBUG_CONFIG="remote_enable=1 remote_mode=req remote_port=9000 remote_host=127.0.0.1 remote_connect_back=0"` for Linux and OS X

Starting a Script with Zend Debugger

Zend Debugger has various configuration options which can be used to let the PHP interpreter reach out to IntelliJ IDEA. These parameters have to be passed to the PHP interpreter using an environment variable:

- `set QUERY_STRING="start_debug=1&debug_host=127.0.0.1&no_remote=1&debug_port=10137&debug_stop=1"` for Windows
- `export QUERY_STRING="start_debug=1&debug_host=127.0.0.1&no_remote=1&debug_port=10137&debug_stop=1"` for Linux and OS X

Configuring Path Mappings

To tell IntelliJ IDEA which path mapping configuration should be used for a connection from a certain machine, set the value of the `PHP_IDE_CONFIG` environment variable to `serverName=SomeName`, where `SomeName` is the name of the debug server configuration defined on the Servers page, see Creating a PHP Debug Server Configuration. Depending on the operating system you are using, set the value in one of the following formats:

- `set PHP_IDE_CONFIG="serverName=SomeName"` for Windows
- `export PHP_IDE_CONFIG="serverName=SomeName"` for Linux and OS X

Troubleshooting

- The debug server configuration is not specified through the `PHP_IDE_CONFIG` environmanet variable. In this case, IntelliJ IDEA detects the host and port from `$amp;_SERVER['SSH_CONNECTION']` and suggests to create a new debug server configuration if it doesn't exists. This works for all connections through ssh even without a tunnel.
- The debug server configuration is not specified through the `PHP_IDE_CONFIG` environment variable and `$amp;_SERVER['SSH_CONNECTION']` is not defined. In this case, a warning is displayed with a link to the instruction in specifying the debug server configuration through the `PHP_IDE_CONFIG` environment variable.
- The debug server configuration is not specified through the `PHP_IDE_CONFIG` environment variable but a wrong format is used, IntelliJ IDEA displays an error message with the instructions.
- The `PHP_IDE_CONFIG` environment variable is configured properly, but the specified debug server configuration does not exist, IntelliJ IDEA displays a warning with a link to the Servers page.

Debugging a PHP HTTP Request

Besides debugging the entire application, you can debug separate HTTP Requests . This is helpful when you are actually interested in a specific page that is accessed in a number of steps, but for this or that reason you cannot specify this page as the start page for debugging, for example, because you need to "come" to this page with certain data.

Debugging PHP HTTP requests in IntelliJ IDEA is supported through the PHP HTTP Request run configuration. Based on the configuration settings, IntelliJ IDEA composes the request to run.

On this page:

- Preparing the debugging engine
- Setting the Breakpoints
- Creating a PHP HTTP Request debug configuration
- Initiating a debugging session and examining the suspended program

Preparing the debugging engine

Before you start debugging, make sure that you have a debugging engine installed and configured properly. IntelliJ IDEA supports debugging with two most popular tools: Xdebug and Zend Debugger . These tools cannot be used simultaneously because they block each other. To avoid this problem, you need to update the corresponding sections in the `php.ini` file as described in Configuring Xdebug and Configuring Zend Debugger .

To open the active `php.ini` file in the editor:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, click … next to the Interpreter field.
3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.

For more information on configuring debugging engines, see Xdebug Installation Guide and Zend Debugger Installation Guide .

Setting the Breakpoints

Breakpoints are source code markers used to trigger actions during a debugging session. Typically, the purpose behind setting a breakpoint is to suspend program execution to allow you to examine program data. However, IntelliJ IDEA can use breakpoints as triggers for a variety of different actions. Breakpoints can be set at any time during the debugging process. Your breakpoints don't affect your source files directly, but the breakpoints and their settings are saved with your IntelliJ IDEA project so you can reuse them across debugging sessions.

1. Place the caret on the desired line of the source code.
   Breakpoints can be set in the PHP context inside `*.php` , `*.html` , and files of other types. Only executable lines are valid locations for line breakpoints. Comments, declarations, and empty lines are not valid locations for the
2. Do one of the following:
   - Click the left gutter area at a line where you want to toggle a breakpoint.
   - On the main menu, choose Run | Toggle Line Breakpoint.
   - Press `Ctrl+F8` .

Creating a PHP HTTP Request debug configuration

IntelliJ IDEA agglutinates the settings specified in this configuration into a PHP HTTP request.

1. Open the Run/Debug Configuration dialog box by doing one of the following:
   - On the main menu, choose Run | Edit Configurations.
   - Press `Shift+Alt+F10` , then press `0` to display the Edit Configuration dialog box or select the configuration from the pop-up window and press `F4` .
2. Click + on the toolbar or press `Insert` . From the drop-down list, select the PHP HTTP Request configuration type. The PHP HTTP Request dialog box opens.
3. Specify the configuration name.
4. In the Server drop-down list, specify the debug server configuration to interact with the Web server where the application is executed. Select one of the existing configurations or click the Browse button … and define a debug server configuration in the Servers dialog box that opens as described in Creating a PHP Debug Server Configuration .
5. In the URL text box, complete the `host` element of the request to debug. Type the path relative to the host specified in the debug server configuration. As you type, IntelliJ IDEA composes the URL address on-the-fly and displays it below the text box.
6. Specify whether you want to bring any data to the target page. From the Request method drop-down list, choose the relevant request type:
   - To access the page without bringing any data, choose GET.
   - To access the page with some data saved in variables, choose POST and type the relevant variables in the Request body text box.
     By default, the Project Encoding is used in requests encoding if it is not specified explicitly, for example:

```
header('Content-type: text/html;charset=utf-8');
```

     The Project Encoding is specified on the File Encodings page, under the Editor node of the Settings / Preferences Dialog .
7. In the Query text box, type the query string of the request. This string will be appended to the request after the `?` symbol.
8. Click OK, when ready.

Initiating a debugging session and examining the suspended program

1. To start debugging, click the Debug button on the toolbar.
2. As soon as the debugger suspends on reaching the first breakpoint, examine the application by analyzing frames. A frame corresponds to an active method or function call and stores the local variables of the called method or function, the arguments to it, and the code context that enables expression evaluation. All currently active frames are displayed on the Frames pane of the Debug tool window . where you can switch between them and analyze the information stored therein in the Variables and Watches panes. For more details, see the section Examining Suspended Program .

3. Continue running the program and examine its frames as soon as it is suspended again.
   - To control the program execution manually, step through the code using the commands under the Run menu or toolbar buttons: Step Into (F7), Step Out (Shift+F8), Step Over (F8), and others. For more details, see Stepping Through the Program .
   - To have the program run automatically up to the next breakpoint, resume the session by choosing Run | Resume Program or pressing F9

Multiuser Debugging via Xdebug Proxies

When using Xdebug, you can debug PHP applications in the multiuser mode via Xdebug proxy servers.

To enable multiuser debugging via an Xdebug proxy server

1. Open the active `php.ini` file in the editor:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. On the PHP page that opens, click … next to the Interpreter field.
   3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.
2. Locate the `xdebug.idekey` setting and assign it a value of your choice.
3. Configure access to the required Xdebug proxy server from your IDE:
   - To register your IDE to the server choose Tools | Xdebug Proxy | Register IDE on the main menu. In the Xdebug Proxy dialog box, that opens, specify the `xdebug.idekey`, the host on which the Xdebug proxy server resides, and the port which IntelliJ IDEA will listen to during a proxy debugging session. When you click OK, IntelliJ IDEA connects to the specified proxy server and sends the specified credentials. The server registers the credentials and sends a confirmation.
   - To update the credentials, choose Tools | Xdebug Proxy | Configuration on the main menu. In the Xdebug Proxy dialog box, that opens, edit the IDE key, the host, and the port settings.
   - To discard the current credentials, choose Tools | Xdebug Proxy | Cancel IDE Registration on the main menu.

Testing PHP Applications

The topics in this part provide guidelines in PHP-specific unit testing procedures. For general information on testing in IntelliJ IDEA, see the section Testing .

IntelliJ IDEA supports unit testing of PHP applications through integration with the PHPUnit tool.

Generally, IntelliJ IDEA runs and debugs PHPUnit tests same way as other applications, by running the run/debug configurations you have created . When doing so, it passes the specified test class, file, or directory to the test runner. You can run unit testing locally and remotely depending on the chosen run configuration.

To create and run unit tests on PHP applications, perform the following general steps:

- Enable PHPUnit support .
- Write the unit tests to run.
- Group the test to distinguish between testing in a production and in your development environment or to enable filtering tests by their authors.
- Create a run configuration :
  - To run unit tests locally, create a PHPUnit configuration.
  - To run unit tests on a remote server, create a PHPUnit by HTTP configuration.
- Launch unit tests by clicking the Run toolbar button and monitor test results in the Run tool window .
- Launch unit tests with coverage by clicking the Run with coverage toolbar button and analyze the test coverage in the Coverage Tool Window .
- Debug unit tests by setting breakpoints where necessary and clicking the Debug toolbar button . For details, see PHP Debugging Session .

Enabling PHPUnit Support

IntelliJ IDEA supports unit testing of PHP applications through integration with the PHPUnit tool.

You can use PHPUnit in IntelliJ IDEA in the following ways:

- Install PHPUnit and run it from PEAR that is configured as include path. The advantage of this approach is that once installed, PHPUnit can be re-used in other IntelliJ IDEA projects.
  The PHPUnit installation procedure depends on the operating system you use and your system settings. Please, refer to the PHPUnit installation instructions for information on installing and configuring this tool.
- Install PHPUnit using the Composer Dependency Manager and run this instance.
- Without installing PHPUnit. Just run it from the `phpunit.phar` archive.

On this page:

- Configuring the PEAR folder with PHPUnit as an include path
- Loading PHPUnit with autoload.php from the Composer Dependency Manager
- Running PHPUnit from the phpunit.phar archive

Configuring the PEAR folder with PHPUnit as an include path

The PHPUnit installation procedure depends on the operating system you use and your system settings. Please, refer to the PHPUnit installation instructions for information on installing and configuring this tool.

1. Open the Settings dialog box, and click PHP. The PHP page opens.
2. From the Interpreter drop-down list, choose the PHP installation to use.
3. Configure the `PEAR` folder as an include path. Click the Add button in the Include path area, and select the `PEAR` folder in the dialog box, that opens.
4. Click PHPUnit under the PHP node, and on the PHPUnit page that opens choose Load from include path.

Loading PHPUnit with autoload.php from the Composer Dependency Manager

The Composer Dependency Manager can download PHPUNit and add it to the current project. The tool loads packages using the `autoload.php` file in the `vendor` folder.

1. Enable Composer in your project. Do one of the following:
   - Download `composer.phar`, configure Composer as a command line tool, and initialize it in your project in the command line mode.
   - Set up Composer in your project through the user interface: choose Tools | Composer | Init Composer and then specify the location of an existing `composer.phar` file or have IntelliJ IDEA download the file. For more details, see Using Composer Dependency Manager.

     When the initialization process is completed, the `autoload.php` file is stored in the `vendor` folder.
2. Open the Settings dialog box, and click PHPUnit under the PHP node.
3. On the PHPUnit page that opens, choose Use custom loader and specify the location of the `autoload.php` in the Path to script text box below.

When you start a testing session for the first time, the `autoload.php` file starts, installs and runs PHPUnit.

Running PHPUnit from the phpunit.phar archive

You can download `phpunit.phar` manually and then tell IntelliJ IDEA to load the libraries from it or have IntelliJ IDEA download `phpunit.phar` automatically. Depending on your preferences, do one of the following:

- Download `phpunit.phar` manually:
  1. Download phpunit.phar manually.
  2. Open the Settings dialog box, and click PHPUnit under the PHP node.
  3. On the PHPUnit page that opens, choose Path to phpunit.phar and specify the location of the archive in the Path to phpunit.phar text box below.
- Have IntelliJ IDEA download `phpunit.phar`:
  1. Open the Settings dialog box, and click PHPUnit under the PHP node.
  2. On the PHPUnit page that opens, choose Path to phpunit.phar and click the Download phpunit.phar... link.
  3. In the dialog box that opens, specify the folder to store the `phpunit.phar` archive after download. IntelliJ IDEA starts the download. After the process is completed successfully the file is stored in the chosen folder and the path to this folder is displayed in the Path to phpunit.phar text box below.

Creating PHPUnit Tests

You can write unit tests manually or have test stubs generated automatically based on the PHP classes that are subject for testing.

A Test class is a PHP class with the name `<production class>Test.php`. For example, if the class to test is `Myclass.php`, the tests for it will be in `MyclassTest.php`.

IntelliJ IDEA also generates tests for classes defined among others within a PHP file. If several production classes are defined in one single file, for each generated test class IntelliJ IDEA will create a separate file.

Test classes are inherited from `PHPUnit_Framework_TestCase`.

The data for generating PHP test classes are specified in the Generate PHPUnit Test dialog box.

On this page:

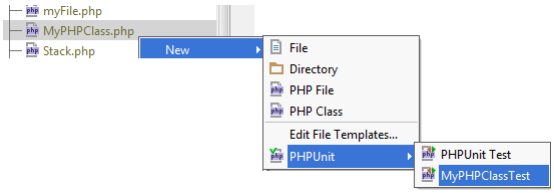
– Prerequisites
– Generating a test for a PHP class defined in a separate file
– Generating a test for a PHP class defined among others within a PHP file
– Generating a PHPUnit test method

Prerequisites

Make sure, the PHPUnit tool is installed on your machine and enabled in IntelliJ IDEA. For more information, see http://www.phpunit.de/manual/current/en/installation.html and Enabling PHPUnit Support.

Generating a test for a PHP class defined in a separate file

1. In the Project tool window, select the PHP class to create unit tests for, e.g. `MyPHPClass` as shown in the image below, and choose New | PHPUnit | PHPUnit Test on the context menu of the selection.



2. In the Generate PHPUnit Test dialog box that opens, specify the following data:
   1. The fully qualified name of the class to be tested in the Class to test area. The specified name will be used to propose the Test Class Name. To use completion, press Control+Space and choose the relevant production class from the pop-up list. By default, the Name text box displays the name of the class on which the test generation was invoked.
   2. The name of the test class. IntelliJ IDEA automatically composes the name from the production class name as follows: `<production class>Test.php`. The test class name is displayed in the Name text box of the Test Class area.
   3. The directory where the file with the test class will be stored. By default, it is the directory where the production class is stored. The default value is displayed in the Directory text box of the Test Class area. To specify another folder, click the Browse button and choose the relevant folder in the dialog box that opens. To use completion, press Control+Space and choose the relevant folder from the pop-up list.
   4. The namespace the test class will belong to. IntelliJ IDEA completes the namespace automatically based on the directory and displays the generated value in the Namespace text box. When the directory is changed, the namespace is changed accordingly. To use completion, press Control+Space and choose the relevant namespace from the pop-up list.
   5. The test file name. By default, the name is displayed in the Name text box and is the same as the test class name.

   Check, accept, or update the predefined settings and click OK to initiate the test generation.

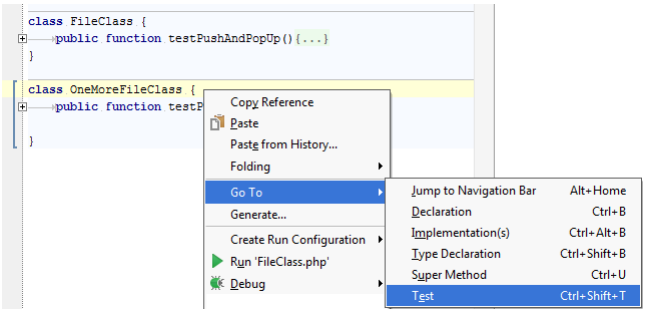
> Tip Alternatively, you select the PHP production class in question, choose New | PHPUnit | PHPUnit Test, and fill in the data manually.

When the test is ready, navigate back to the production class by choosing Navigate | Go to Test Subject. For details, see Navigating Between Test and Test Subject.

Generating a test for a PHP class defined among others within a PHP file

1. In the editor, open the file with the class to generate the test for.

2. Do one of the following:
   – On the main menu, choose File | New | PHPUnit | PHPUnit Test.
   – On the context menu, choose Go To | Test on the context menu, then choose Create New Test in the pop-up list.



3. In the Generate PHPUnit Test dialog box that opens, specify the following data:
   1. The fully qualified name of the class to be tested in the Class to test area. To use completion, press Control+Space and choose the relevant production

class from the pop-up list. The specified name will be used to propose the Test Class Name.

2. The name of the test class. IntelliJ IDEA automatically generates the name and displays it in the Name text box of the Test Class area.
3. The directory where the file with the test class will be stored. By default, it is the directory where the production class is stored. The default value is displayed in the Directory text box of the Test Class area. To specify another folder, click the Browse button and choose the relevant folder in the dialog box that opens. To use completion, press Control+Space and choose the relevant folder from the pop-up list.
4. The namespace the test class will belong to. IntelliJ IDEA completes the namespace automatically based on the directory and displays the generated value in the Namespace text box. When the directory is changed, the namespace is changed accordingly. To use completion, press Control+Space and choose the relevant namespace from the pop-up list.
5. The test file name. By default, the name is displayed in the Name text box and is the same as the test class name.

Check, accept, or update the predefined settings and click OK to initiate the test generation.

Generating a PHPUnit test method

Besides generating test class stubs, IntelliJ IDEA can help you populate them by generating stubs for test methods.

1. Open the required test class in the editor, position the cursor anywhere inside the class definition, and do one of the following:
   - On the main menu, choose Code | Generate, then choose PHPUnit Test Method from the Generate pop-up list.
   - On the context menu, choose Generate, then choose PHPUnit Test Method from the Generate pop-up list.

2. Set up the test fixture, that is, have IntelliJ IDEA generate stubs for the code that emulates the required environment before test start and returns the original environment after the test is over. For more details, see http://phpunit.de/manual/3.7/en/fixtures.html .
   1. On the context menu, choose Generate | Override method.
   2. From the Choose methods to override dialog box that opens, choose SetUp or TearDown and click OK.

Grouping Tests

Grouping tests is helpful, for example, to distinguish between tests to run in a production environment from those to run in your development environment. You just need to create two groups and then include or exclude them depending on the current environment.

In this section:

- Attaching a test to a group
- Filtering tests by their authors
- Including and excluding test groups from execution

To attach a test to a group

- Tag the test with the @group annotation in the format:

```
/**
@group <group specification>
*/
```

To enable filtering a test by its author

- Tag the test with the @author annotation in the format:

```
/**
@author <author specification>
*/
```

To configure execution of a test group

1. In the Run/Debug Configuration: PHPUnit by HTTP dialog box, open the Test groups tab.
2. To enable appointing groups to execute, select the Include/Exclude check box.
3. In the Group Name list, select the relevant group and do one of the following:
   - To have the tests in the selected group executed, select the Include check box.
   - To have the tests in the selected group skipped, select the Exclude check box.

Monitoring Code Coverage for PHP Applications

Code Coverage for PHPUnit shows you how much of your code is covered with tests and marks covered and uncovered lines visually right in the editor.

In this section:

– Prerequisites
– Configuring code coverage
– Measuring code coverage

Prerequisites

1. The PHPUnit tool is installed on your machine and enabled in IntelliJ IDEA .
2. A Xdebug or Zend Debugger debugging engine is configured in the current PHP interpreter.

To configure code coverage

1. Open the Settings dialog box .
2. Specify coverage options on the Coverage page.

To measure code coverage

1. Write the tests manually.
2. Create a run configuration of the type PHPUnit .
3. On the main toolbar, select the PHPUnit run configuration in the Run/Debug Configurations drop-down list and click the Run with Coverage button .
4. Monitor the code coverage in the Coverage tool window.

Using Behat Framework

With IntelliJ IDEA, you can practise behaviour-driven development by running scenarios using the BehatBehat framework. Currently IntelliJ IDEA supports integration with Behat 3 and Behat 2 versions.

Native support of Behat in IntelliJ IDEA includes:

- Recognition of and coding assistance for `.feature` scenario files and `.php` scenario definition files.
- Support of Gherkin syntax in `.feature` files: Feature, Scenario, Given, When, Then, And, and But keywords.
- Recognition of @given, @when, and @then annotations in definition files.
- Setting correspondence between scenarios and their definitions through regular expressions in accordance with the PCRE standard for Behat 2.4 and PCRE+ for Behat 3.0. Turnip expressions are also welcome.

On this page:

- Before you start
- Downloading and installing Behat
  - Downloading the behat.phar Archive
  - Installing Behat using the Composer dependency manager
- Registering the Behat installation in IntelliJ IDEA
- Creating a Run Configuration
- Launching Tests

Before you start

1. Make sure the PHP interpreter is configured in IntelliJ IDEA on the PHP page , as described in Enabling PHP Support .
2. Make sure the PHP and Behat plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.
3. Configure the PHP interpreter to use Behat with, as described Configuring Remote PHP Interpreters . Note that Behat 3 requires PHP 5.5 and higher.

Downloading and installing Behat

You can acquire Behat in the following ways:

- Download the `behat.phar` archive.
- Use the Composer dependency manager.

Downloading the behat.phar Archive

1. Download `behat.phar` at the Behat Downloads page .
2. Store `behat.phar` on your computer:
   - To get full coding assistance in addition to simply running Behat scenarios, store `behat.phar` under the root of the project where Behat will be later used. In this case, IntelliJ IDEA will include it in indexing, so references to Behat classes will be successfully resolved and you will be able not only to run scenarios but also provided with full coding assistance.
   - If you only need to run Behat scenarios and you do not need any coding assistance, you can save `behat.phar` outside the project. In this case, IntelliJ IDEA will only run Behat scenarios without providing coding assistance.

Installing Behat using the Composer dependency manager

Before you start, make sure Composer is installed on your machine and initialized in the current project, see Using Composer Dependency Manager .

1. On the main menu, choose Tools | Composer | Add Dependency.
2. In the Add Composer Dependency dialog that opens, select the `behat/behat` package from the Available Packages list, possibly using the search field. The list shows all the available packages, however, the packages that have been already installed are marked with a tick.
   Choose the relevant version from the Version to install drop-down list.
3. If necessary, expand the Settings hidden area and specify the advanced installation options. In the Command line parameters text box, type the additional command line parameters to adjust the add dependency procedure. For example, to have the package added to the `require-dev` section instead of the default `require` section type `-dev` . For more information about Composer command line options during installation, see https://getcomposer.org/doc/03-cli.md .
4. Click Install.

When you click Finish, the `create-project` command is invoked with the selected package. This results in creating a Composer project whose configuration and structure depends on the selected package, see https://getcomposer.org/doc/03-cli.md#create-project for details. After that a IntelliJ IDEA project opens.

Registering the Behat installation in IntelliJ IDEA

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Behat under PHP.
2. On the Behat that opens, specify the location of the Behat executable file or `behat.phar` archive and the PHP interpreter to use Behat with. This PHP interpreter can be installed either locally on your computer, or in a Vagrant virtual environment, or on a remote host. For information on configuring PHP interpreters, see Configuring Local PHP Interpreters and Configuring Remote PHP Interpreters .
   Click the Add toolbar button + in the central panel and do one of the following depending on your workflow:
   - To use Behat with a local PHP interpreter:
     1. Choose Local.
     2. In the Behat Library area, specify the location of the Behat executable file or `behat.phar` archive.
     3. Click next to the Path to Behat directory or phar file text box. IntelliJ IDEA detects the version of Behat and displays it below the text box.
   - To use Behat in a Vagrant environment:
     1. Choose By Remote Interpreter. In the Behat by Remote Interpreter dialog box, choose the desired PHP interpreter configuration in a Vagrant environment. When you click OK, IntelliJ IDEA returns to the Behat page.

IntelliJ IDEA displays the appointed PHP interpreter configuration in the Interpreter field. The field is read-only, to update the chosen interpreter configuration or create a new one, click the ... whereupon IntelliJ IDEA brings you to the Interpreters dialog box .

2. In the Behat Library area, specify the location of the Behat executable file or `behat.phar` archive in the target Vagrant virtual environment. For example, if you installed Behat through the Composer dependency manager, the location is `vendor/bin/behat` .
3. Click next to the Path to Behat directory or phar file text box. IntelliJ IDEA detects the version of Behat and displays it below the text box.

– To use Behat on a remote host with SSH access:

1. Choose By Remote Interpreter. In the Behat by Remote Interpreter dialog box, choose the desired PHP interpreter configuration on a remote host. When you click OK, IntelliJ IDEA returns to the Behat page.
   IntelliJ IDEA displays the appointed PHP interpreter configuration in the Interpreter field. The field is read-only, to update the chosen interpreter configuration or create a new one, click the ... whereupon IntelliJ IDEA brings you to the Interpreters dialog box .
2. In the Behat Library area, specify the location of the Behat executable file or `behat.phar` archive on the target host.
3. Click next to the Path to Behat directory or phar file text box. IntelliJ IDEA detects the version of Behat and displays it below the text box.

3. In the Test Runner area, appoint the configuration `.yaml` file to use for launching and executing scenarios.
   By default, Behat looks for a `behat.yaml` configuration file in the project root folder or in the `config` folder. You can appoint a custom configuration file.
   – To have Behat use the `behat.yaml` configuration file from the project root folder or from the `config` folder. If no such file is found, test execution fails, therefore it may be more reliable to specify the configuration file explicitly. clear the Default configuration file check box.
   – To specify your own `.yaml` configuration file. Ths file will be later used as default in all Behat run/debug configurations.
   In the text box, specify the location of the configuration file to use. Type the path manually or click the ... button and choose the file in the dialog box that opens.

   select the Default configuration file check box.

Creating a Run Configuration

1. Choose Run | Edit Configuration on the main menu. Alternatively, click `Shift+Alt+F10` and select Edit Configuration from the pop-up menu.
2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button +, and choose Behat on the context menu.
3. In the Run/Debug Configuration: Behat dialog box , specify the scenarios to run and customize the behavior of the current PHP interpreter by specifying the options and arguments to be passed to the PHP executable file.

Launching Tests

For information about writing Behat features, see http://docs.behat.org/en/latest/guides/1.gherkin.html . To run or debug your tests, do one of the following:

– On the main menu, choose Run | Run <configuration name> and monitor test results in the Run tool window as described in Viewing and Exploring Test Results .

– On the main menu, choose Run | Debug <configuration name> and debug the tests in the Debug tool window .

Profiling the Performance of a PHP Application

Besides pure debugging, you can also profile the performance of your applications. IntelliJ IDEA provides visual representation of the profiling data collected by the debugging engine you are currently using.

In this section:

- Profiling with Xdebug
  - Enabling Profiling with Xdebug
  - Analyzing Xdebug Profiling Data
- Profiling with Zend Debugger
  - Enabling Profiling with Zend Debugger
  - Analyzing Zend Debugger Profiling Data

Profiling with Xdebug

Besides interactive debugging , IntelliJ IDEA integration with Xdebug also supports profiling. IntelliJ IDEA provides visual representation of profiling data generated by Xdebug. You can select several snapshots at a time and collect the aggregated profiling information.

Before profiling with Xdebug, download, install and configure the components of the PHP development environment . Normally, these are a PHP engine, a web server, and the Xdebug tool.

In this part:

– Enabling Profiling with Xdebug

– Analyzing Xdebug Profiling Data

Enabling Profiling with Xdebug

Xdebug profiler is incorporated in the Xdebug tool. Therefore you first need to download, install, and enable Xdebug itself and after that enable the profiling functionality within it.

To enable profiling with Xdebug, perform these general steps:

- Configuring the Xdebug Tool
- Enabling the Xdebug Profiler
- Configuring the Way to Toggle the Profiler from the Browser
- Specifying the location for storing accumulated profiling data

Configuring the Xdebug Tool

1. Download and install the Xdebug tool.
2. Integrate Xdebug with the PHP engine .
3. Integrate Xdebug with IntelliJ IDEA .

Enabling the Xdebug Profiler

1. Open the active `php.ini` file in the editor:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. On the PHP page that opens, click ... next to the Interpreter field.
   3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.
2. Set the xdebug.profiler_enable directive to 1:

```
xdebug.profiler_enable = 1;
```

3. To enable toggling the profiler from the browser through control over debugger cookies, set the xdebug.profiler_enable_trigger directive to 1:

```
xdebug.profiler_enable_trigger = 1;
```

Configuring the Way to Toggle the Profiler from the Browser

To specify the GET/POST or COOKIE parameters, do one of the following:

- Specify the values manually .
- Generate the bookmarklets to toggle the debugger through. These bookmarklets will appear on the toolbar of your browser. They provide control over the debugger cookie, through them you will activate and deactivate the debugger.
  1. Enable the Bookmarks toolbar in your browser by doing one of the following depending on the browser type:
     - In Firefox, choose View | Toolbar | Bookmarks Toolbar.
     - In Chrome, choose Bookmarks | Show bookmarks bar.
  2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Debug under PHP.
  3. On the Debug page, that opens, click the Use debugger bookmarklets to initiate debugger from your favorite browser link.
  4. On the Zend Debugger & Xdebug bookmarklets page that opens, check the debugging engine settings and click Generate. The bookmarks for listed debugging-related actions are generated.
  5. Drag the generated links to the bookmark toolbar in your browser.

Specifying the location for storing accumulated profiling data

1. Open the active `php.ini` file in the editor:
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. On the PHP page that opens, click ... next to the Interpreter field.
   3. In the Interpreters dialog box that opens, the Configuration File read-only field shows the path to the active `php.ini` file. Click Open in Editor.
2. Define location for accumulating profiling snapshots by specifying the xdebug.profiler_output_dir directive.

```
xdebug.profiler_output_dir = "<output folder name>"
```

3. Specify the name of the file to store snapshots in through the value of the `xdebug.profiler_output_name` directive. The default name is `cachegrind.out.%p`, where `%p` is the name format specifier. Accept the default name or define a custom one in compliance with the following standard:
   1. The name should always be `cachegrind.out` .
   2. Use the supported format specifiers .

Analyzing Xdebug Profiling Data

When integration with Xdebug profiler is enabled , IntelliJ IDEA provides visual representation of profiler snapshots. IntelliJ IDEA opens a separate editor tab with four views where the data are presented based on different criteria.

To have the profiling data collected and analyze it, perform these general steps:

- Initiating an Xdebug Debugging Session
- Retrieving the Data Accumulated by the Profiler
- Examining the Profiling Data

Initiating an Xdebug Debugging Session

To start an Xdebug session, do one of the following:

- To start debugging an entire application, create debug configuration of the type PHP Web Application, and launch debugging by clicking the Debug toolbar button. See Debugging with a PHP Web Application Debug Configuration for details.
- To debug a specific PHP HTTP request, define a debug configuration of the type PHP HTTP Request, and launch debugging by clicking the Debug toolbar button. See Debugging a PHP HTTP Request for details.
- To initiate a zero-configuration debugging session:
  1. Toggle the Start Listen PHP Debug Connections button so it changes to . After that IntelliJ IDEA starts listening to the port of the debugging engine used in the current project . Ports for debuggers are set at the IntelliJ IDEA level in the Debug dialog box (File | Settings | PHP | Debug).
  2. Open the starting page of your application in the browser, choose the Start debugger bookmark to activate the debugging engine from the browser, re-load the current page (the starting page of the application), and then return to IntelliJ IDEA.

Retrieving the Data Accumulated by the Profiler

1. On the main menu, choose Tools | Analyze Xdebug Profiler Snapshot.
2. In the Select Xdebug profiler snapshot dialog box, that opens, choose the folder and the file where the profiling data is stored .
   IntelliJ IDEA presents the collected profiling data in a separate editor tab with the name of the selected profiler output file.

Examining the Profiling Data

When you request on the accumulated profiling data, IntelliJ IDEA opens its visualized presentation in a separate editor tab. The tab is named after the selected profiler output file and consists of several views. Switch between the views to analyze the profiling data based on various criteria of analysis.

- In the Execution Statistics view, examine the summary information about execution metrics of every called function.
- In the Call Tree view, explore the execution paths of all called functions.
- To explore the execution paths of a specific function, select the function in question in the Call Tree view and view its callees in the Callees view.
- To explore all the paths that can result in calling a specific function, select the function in question in the Call Tree view and examine its possible callers in the Callers view.

Profiling with Zend Debugger

Besides interactive debugging , IntelliJ IDEA integration with Xdebug also supports profiling. IntelliJ IDEA provides visual representation of profiling data generated by Zend Debugger.

Before profiling with Zend Debugger, download, install and configure the components of the PHP development environment . Normally, these are a PHP engine, a web server, and the Zend Debugger tool.

Enabling Profiling with Zend Debugger

Xdebug profiler is incorporated in the Zend Debugger tool. Therefore you only need to download, install, and enable Xdebug itself.

To enable profiling with Zend Debugger, perform these general steps:

- Configuring Zend Debugger
- Configuring the Way to Toggle the Profiler from the Browser

Configuring Zend Debugger

1. Download and install the Zend Debugger tool.
2. Integrate Zend Debugger with the PHP engine .
3. Integrate Xdebug with IntelliJ IDEA .

Configuring the Way to Toggle the Profiler from the Browser

To specify the GET/POST or COOKIE parameters, do one of the following:

- Generate the bookmarklets to toggle the debugger through. These bookmarklets will appear on the toolbar of your browser. They provide control over the debugger cookie, through them you will activate and deactivate the debugger.
  1. Enable the Bookmarks toolbar in your browser by doing one of the following depending on the browser type:
     - In Firefox, choose View | Toolbar | Bookmarks Toolbar.
     - In Chrome, choose Bookmarks | Show bookmarks bar.
  2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Debug under PHP.
  3. On the Debug page, that opens, click the Use debugger bookmarklets to initiate debugger from your favorite browser link.
  4. On the Zend Debugger & Xdebug bookmarklets page that opens, check the debugging engine settings and click Generate. The bookmarks for listed debugging-related actions are generated.
  5. Drag the generated links to the bookmark toolbar in your browser.
- Specify the values manually.

Analyzing Zend Debugger Profiling Data

When integration with the Zend Debugger profiler is enabled , IntelliJ IDEA provides visual representation of profiler snapshots. IntelliJ IDEA opens a separate editor tab with four views where the data are presented based on different criteria.

With Zend Debugger, profiling is supported within a zero configuration debugging session .

To have the profiling data collected and analyze it, perform these general steps:

- Initiating a Zero Configuration Zend Debugger Session
- Examining the profiling data

Initiating a Zero Configuration Zend Debugger Session

1. Generate the bookmarklets to toggle the debugger through. These bookmarklets will appear on the toolbar of your browser. They provide control over the debugger cookie, through them you will activate and deactivate the debugger.
   1. Enable the Bookmarks toolbar in your browser by doing one of the following depending on the browser type:
      - In Firefox, choose View | Toolbar | Bookmarks Toolbar.
      - In Chrome, choose Bookmarks | Show bookmarks bar.
   2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Debug under PHP.
   3. On the Debug page, that opens, click the Use debugger bookmarklets to initiate debugger from your favorite browser link.
   4. On the Zend Debugger & Xdebug bookmarklets page that opens, check the debugging engine settings and click Generate. The bookmarks for listed debugging-related actions are generated.
   5. Drag the generated links to the bookmark toolbar in your browser.
2. Toggle the Start Listen PHP Debug Connections button so it changes to . After that IntelliJ IDEA starts listening to the port of the debugging engine used in the current project . Ports for debuggers are set at the IntelliJ IDEA level in the Debug dialog box (File | Settings | PHP | Debug).
3. Open the starting page of your application in the browser, choose the Start debugger bookmark to activate the debugging engine from the browser, re-load the current page (the starting page of the application), and then return to IntelliJ IDEA.
   Establishing a Zero-Configuration debugging session may fail, with no breakpoints hit and therefore the script not suspended. This may happen if if the path mappings are not configured or configured erroneously, or if you have not set any breakpoints. In the latter case, enabling selecting the Break at First Line in PHP Scripts check box in the External Connections area or turning the Run | Break at First Line in PHP Scripts option on the main menu may also help.

   To have IntelliJ IDEA display a notification if the script is not suspended, select the Notify if debug session was finished without being stopped check box in the Advanced Settings area on the Debug page of the Settings dialog box.
4. In the dialog box, that opens, select the incoming connection to profile and click Accept. The Incoming Connection from Zend Debugger dialog box appears only once, when you accept connection from this host for the first time.
   IntelliJ IDEA presents the collected profiling data in a separate editor tab with the name of the selected profiler output file.

Examining the profiling data

When you request on the accumulated profiling data, IntelliJ IDEA opens its visualized presentation in a separate editor tab. The tab is named after the file that implements the page you are currently profiling and consists of several views. Switch between the views to analyze the profiling data based on various criteria of analysis.

- In the Execution Statistics view, examine the summary information about execution metrics of every called function.
- In the Call Tree view, explore the execution paths of all called functions.
- To explore the execution paths of a specific function, select the function in question in the Call Tree view and view its callees in the Callees view.
- To explore all the paths that can result in calling a specific function, select the function in question in the Call Tree view and examine its possible callers in the Callers view.

PHP-Specific Command Line Tools

With IntelliJ IDEA, you can use PHP from the command line through integration with PHP-specific command line tools. Integration with command line tools is provided at the IntelliJ IDEA level, so once configured, a tool can be used in all your IntelliJ IDEA projects. Just activate or de-activate it when necessary depending on the needs of each specific project.

IntelliJ IDEA supports integration with third-party PHP-specific tools:

1. Symfony, version 1.1 or higher.
2. Symfony2 that incorporates the Twig template engine. IntelliJ IDEA supports the official Symfony2 code style for Twig and automatically inserts one space after an opening pair of curvy braces and before a closing pair of curvy braces in Twig templates:

```
{{ some_variable }}
```

3. Zend Framework.
4. Yii.

Alternatively, you can create your own tool and populate it with commands that fit your preferences and needs. The available tools are listed in the Command Line Tool Support page of the Settings dialog box.

To work with a PHP command line tool, perform these general steps:

- Download and enable the required third party tool or define a custom tool.
- Create and open a Symfony or a Yii project, open it in IntelliJ IDEA, and monitor its structure in the Framework tool window.
- Customize the tool commands, if necessary, and have the tool definition file validated.
- Reload the commands from the executable file.
  This functionality is currently available for the Symfony tool only. You can add new tasks to the command definition file and enable IntelliJ IDEA to use them without re-configuring the command line tool, that is, without removing it from the list of available ones and adding it anew.
- Invoke commands and analyze the results of their execution in the dedicated Command Line Tools Console tool window.

Enabling a Command Line Tool

The PHP command line tools that are available from IntelliJ IDEA are listed in the Command Line Tool Support page of the Settings dialog box. Integration with command line tools is provided at the IntelliJ IDEA level, so once configured, a tool can be used in all your IntelliJ IDEA projects. Just activate or de-activate it when necessary depending on the needs of each specific project.

You can download and activate an existing third-party tool or create your own tool . You can update the command definitions of third-party or a custom tool right in the IntelliJ IDEA editor.

In this section:

- Enabling integration with a third party PHP command line tool
- Creating a custom PHP command line tool

To enable integration with a third party PHP command line tool

Integration with command line tools is supported via the Command Line Tool Support plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

1. Download and install the desired tool. Currently IntelliJ IDEA supports integration with the following tools:
   1. Symfony version 1.1 or higher.
   2. Zend Framework 1, see http://framework.zend.com/download/overview .
   3. Zend Framework 2 (ZFTool, see http://framework.zend.com/downloads/latest .
   4. Composer Dependency Manager .
   5. Drush , version 5.8 and higher.
   6. Symfony console-based tools Laravel and Doctrine . These tools are installed through Composer.
2. Open the IntelliJ IDEA settings and click Command Line Tool Support.
3. In the Command Line Tool Support page, click the Add button. In the Choose Tool to Add dialog box that opens, choose Zend Framework or Symfony.
4. In the dialog box that opens, specify the path to the definition file of the chosen tool in the Path to <tool> text box. Depending on the chosen tool, specify the location of the following file:
   - `<Zend-Framework-home>/zf.bat` for Zend Framework 1.
   - `<Zend-Framework-home>/zf.php` or `<Zend-Framework-home>/zf.php` for Zend Framework 2.
   - `<Symfony-home>/data/bin/symfony` for Symfony 1.*.
   - `<Symfony-home>/app/console` for Symfony 2.
   - `<Composer-home>/composer.phar` for Composer
   - `C:/ProgramData/Drush/drush.bat` on Windows and `/usr/bin/drush` on Mac OS or Linux for Drush.
   - For Laravel, the path to the artisan bundled with the tool, for example, `C:\MY_PROJECTS\PHP_PROJECTS\MyLaravelProject\artisan`.
   - For Doctrine, the path to the tool home.

   IntelliJ IDEA searches the contents of the specified file for definitions of commands. When the file analyzes is completed, IntelliJ IDEA displays the specified file in the list.
5. In the Alias text box, specify the alias to use in calls of tool commands. Accept the default alias or edit it, if necessary.
6. To activate the detected command set, select the Enable check box.
7. Customize the command set , if necessary.
8. In the Show console in area, specify where you want the Input pane for typing commands opened:
   - To have the Input pane opened in a pop-up window, choose the Pop-up option.
   - To have the Input pane opened as a text box at the bottom of the Command Line Tools Console tool window, choose the Tool window option.

To create a custom command line tool

1. Open the IntelliJ IDEA settings and click Command Line Tool Support.
2. In the Command Line Tool Support page, click . In the Command Line Tools dialog box that opens, choose visibility level of the tool (project or global), and its type from the Choose tool drop-down list.
3. In the dialog box that opens, specify tool name and location, and the other required parameters.
   When you click OK, IntelliJ IDEA brings you to the Command Line Tool Support page, where the new tool is added to the list of available tools.
4. Select the newly created tool and click . In the tool definition `.xml` file that opens in the editor, define the tool commands.
5. In the Show console in area, specify where you want the Input pane for typing commands opened:
   - To have the Input pane opened in a pop-up window, choose the Pop-up option.
   - To have the Input pane opened as a text box at the bottom of the Command Line Tools Console tool window, choose the Tool window option.

Using Framework Model View Controller

With IntelliJ IDEA, you can examine the structure of a project in the terms of the Model-View-Controller pattern . This functionality is provided through in the dedicated Framework tool window and is currently available for the Symfony2 and Yii command line tools.

In this section:

– Prerequisites
– Enabling the MVC view
– Examining the MVC structure of a tool-specific project

Prerequisites

1. The Command Line Tool Support and Framework MVC Structure Support plugins are downloaded, installed, and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.
2. The Symfony2 or Yii command line tool is downloaded.
3. Integration with the tool is enabled on the Command Line Tool Support page of the Settings dialog box.
4. A Symfony2 or a Yii specific project is available on your machine.

To enable the MVC view

1. Open the IntelliJ IDEA settings and click Framework Integration.
2. In the Framework Integration page, specify the command line tool your project is based on and the location of the tool project. IntelliJ IDEA checks the contents of the specified tool project directory and accepts the appointed project root if it contains the following subfolders:
   – For Symfony2: `app` , `src` , `vendor` , and `web` subfolders.
   – For Yii: a `protected` subfolder.

After that, the Framework tool window is available through the View | Tool Windows menu.

To examine the MVC structure of a tool-specific project

1. To open the required tool-specific project, choose File | Open or File | Open Directory, and then select the project root folder in the dialog box that opens.
2. In the Framework tool window that immediately opens automatically, examine the project in the terms of bundles, controllers, and other elements of the MVC pattern.

Using Phing

IntelliJ IDEA supports integration with the Phing build framework. Phing is mainly intended for building PHP projects, but it can be also used as a build tool in a number of areas. Phing functionality in IntelliJ IDEA is provided through a dedicated Phing Build tool window.

In this section:

- Before You Start
- Accessing the Phing Build tool window

Before You Start

1. Make sure Phing is downloaded and set up on your computer. If you are using an AMP , the package may already contain Phing.
2. Install and enable the Phing Support plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Accessing the Phing Build tool window

- Open a Phing build file in the editor or select it in the Project tool window, and then choose Add as Phing build file on the context menu of the selection.
- Choose View | Tool Windows | Phing Build on the main menu. The tool window can be accessed after you have opened it through the context menu of a Phing build file in the editor or in the Project tool window.

Enabling Phing Support

At the project level, you can enable Phing in the following ways:

- Explicitly through the project settings, on the Phing page of the Settings dialog box.
- When adding a file to the list of Phing build files. A correct Phing build file is an `xml` file with the root element `<project>`.
- On the fly, when you attempt to run a build from the editor or from the project tool window.

In either case, the Phing Build tool window becomes available from the View | Tool Windows | Phing Build menu within the scope of the current project.

On this page:

- Before You Start
- Enabling Phing support through the project settings
- Enabling Phing support from the Phing Build tool window
- Enabling Phing on the fly, when attempting to run a build file

Before You Start

1. Make sure Phing is downloaded and set up on your computer. If you are using an AMP, the package may already contain Phing.
2. Install and enable the Phing Support plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins.

Enabling Phing support through the project settings

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Phing under PHP.
2. In the Phing page that opens, specify the location of the file `phing.bat` in the Path to Phing executable text box. Type the path manually or click the Browse button ... and choose the file location in the dialog box that opens.

Enabling Phing support from the Phing Build tool window

1. Open a Phing build file in the editor or select it in the Project tool window.

   Tip A correct Phing build file is an `xml` file with the root element `<project>`.
2. On the context menu of the editor or the selection, choose Add as Phing build file.
3. In the Phing Build tool window, that opens, click the Settings button on the toolbar.
   The button is only available if the list of build files is not empty.
4. In the Phing Settings dialog box, that opens, specify the location of the `phing.bat` executable file.

Enabling Phing on the fly, when attempting to run a build file

1. Open an Phing build file in the editor.
2. On the context menu of the editor, choose Run Build File.
3. In the Phing Build tool window, that opens, click the Settings button on the toolbar.
   The button is only available if the list of build files is not empty.
4. In the Phing Settings dialog box, that opens, specify the location of the `phing.bat` executable file.

Creating a List of Phing Build Files

A correct Phing build file is an `xml` file with the root element `<project>`. Build files are created outside the Phing Build tool window and are normally stored under your PHP project root folder.. Find more about writing Phing build files in Phing Getting Started .

In this section:

– Accessing the Phing Build tool window
– Managing lists of build files

Accessing the Phing Build tool window

– Open a Phing build file in the editor or select it in the Project tool window, and then choose Add as Phing build file on the context menu of the selection.

– Choose View | Tool Windows | Phing Build on the main menu. The tool window can be accessed after you have opened it through the context menu of a Phing build file in the editor or in the Project tool window.

To configure a list of build files, perform these general steps

– Open the Phing Build tool window.

– To add a build file to the list, click the Add button + on the toolbar and choose the required `.xml` build file in the Select Phing Build File dialog box, that opens.

– To remove a build file from the list, select the file and click the Remove button – on the toolbar.

– To navigate to the source code of a build file, select the desired file and choose Jump to Source on the context menu of the selection.

Managing Phing Build Targets

Phing functionality in IntelliJ IDEA is provided through a dedicated Phing Build tool window.

In this section:

- Accessing the Phing Build tool window
- Examining the build targets defined in a build file
- Hiding targets
- Associating a shortcut with a Phing target
- Appointing targets for execution before running or debugging

Accessing the Phing Build tool window

- Open a Phing build file in the editor or select it in the Project tool window, and then choose Add as Phing build file on the context menu of the selection.
- Choose View | Tool Windows | Phing Build on the main menu. The tool window can be accessed after you have opened it through the context menu of a Phing build file in the editor or in the Project tool window.

Examining the build targets defined in a build file

- Open the Phing Build tool window. The window shows all the build files as nodes.
- To view the build targets defined in a specific build file, expand the corresponding node by clicking next to the build file name. Note that this expanding does not affect the targets marked as hidden.
- To fold or unfold the build targets defined in all build files, click respectively the Expand All button or the Collapse All button on the toolbar. Note, that unfolding does not affect the targets marked as hidden.
- To navigate to the definition of a target in the source code, select the desired target and choose Jump to Source on the context menu of the selection.

Hiding targets

Among the targets defined in a build file, you may have some that are only called by other targets and are never run alone. You can suppress showing such targets in the build file tree by marking them as hidden.

Hidden targets do not become visible when you expand the node of a specific build file or click Expand All button .

You can mark a target as hidden both directly from the Phing Build tool window or from the Phing Settings dialog box. However, the hidden status can be removed only through the Phing Settings dialog box.

- To mark one or several build targets as hidden, do one of the following:
  - In the Phing Build tool window, select the targets under their build file node and choose Mark to Hide on the context menu of the selection.
  - Select the build file in which they are defined and click the Settings button on the toolbar. Then in the Phing Settings dialog box that opens, switch to the Hiding targets tab and select the Hide check boxes next to the targets to be hidden.
- To remove the hidden status of a target:
  1. Select the build file in which it is defined and click the Settings button on the toolbar.
  2. In the Phing Settings dialog box that opens, switch to the Hiding targets tab where the Hide check boxes next to the names of the hidden targets are selected. Clear the Hide check boxes next to the targets for which you want the status hidden removed.

Associating a shortcut with a Phing target

You can associate a build target with a keyboard shortcut and execute commonly-used targets with a single key-stroke. If a Phing build file is added to the project, its targets appear under the Phing Targets node in the Keymap dialog box.

1. In the Phing Build tool window, right-click the desired build target.
2. On the context menu, choose Assign Shortcut.
3. In the Keymap dialog box that opens, configure the shortcut as described in Configuring Keyboard Shortcuts .

Appointing targets for execution before running or debugging

1. Open the Phing Build tool window.
2. Select the desired target and choose Before Run/Debug on the context menu of the selection.
3. In the Execute Target Before Run/Debug dialog box that opens, select the configurations before which you want the target executed.

Running Builds

Phing functionality in IntelliJ IDEA is provided through a dedicated Phing Build tool window.

In this section:

- Accessing the Phing Build tool window
- Running builds
- Running separate build targets

Accessing the Phing Build tool window

- Open a Phing build file in the editor or select it in the Project tool window, and then choose Add as Phing build file on the context menu of the selection.
- Choose View | Tool Windows | Phing Build on the main menu. The tool window can be accessed after you have opened it through the context menu of a Phing build file in the editor or in the Project tool window.

To run a build file

1. Open the Phing Build tool window.
2. Select the required build file in the list and click the Run button on the toolbar or choose Run Build on the context menu of the selection.

To run a build target, do one of the following:

- In the Phing Build tool window, select the required build target in the list and click the Run button on the toolbar or choose Run Target on the context menu of the selection.
- Use the shortcut associated with the target .

Customizing Build Execution by Configuring Properties Externally

A Phing build file normally contains property elements that can be configured either manually in the build file itself or externally, in the Phing Build tool window. Properties that are configured externally are passed to Phing through a command line when build execution is launched and therefore they are every time re-calculated dynamically. Externally configured properties always override the ones set manually in the build file.

In this section:

- Accessing the Phing Build tool window
- Configuring a property externally

Accessing the Phing Build tool window

- Open a Phing build file in the editor or select it in the Project tool window, and then choose Add as Phing build file on the context menu of the selection.
- Choose View | Tool Windows | Phing Build on the main menu. The tool window can be accessed after you have opened it through the context menu of a Phing build file in the editor or in the Project tool window.

To configure a property externally

1. Open the Phing Build tool window.
2. Select the required build file in the list and click the Settings button on the toolbar.
3. In the Phing Settings dialog box that opens, switch to the Properties tab and configure a list or properties to be passed to Phing through a command line.
   1. To add a new property to the list, click the Add button.
   2. In the Property text box, type the name of the property.
   3. In the Value text box, specify the property value to be passed. Do one of the following:
      - Type the value manually.
      - To use IntelliJ IDEA macros , click the Insert macro button and configure a list of relevant macro definitions in the Macros dialog box that opens. To add a macro, select it in the list and click Insert.
4. When the list of properties is ready, click OK in the Phing Settings dialog box.

Using Composer Dependency Manager

The Composer Dependency Manager is very helpful when you need to install PHP-related frameworks, for example PHPUnit or Behat test frameworks.

You can use Composer from IntelliJ IDEA in three modes:

- Using the dedicated IntelliJ IDEA interface for most basic operations such as initializing a project with a `composer.json` stub created and adding dependencies .
- To have all Composer commands at disposal, you need to configure Composer as a command line tool and use it in the command line mode from the dedicated Command Line Tools Console Tool Window or pop-up window. For more details, see Running Command Line Tool Commands .
- You can have IntelliJ IDEA generate a project stub from an existing package using Composer. In this case, Composer will be initialized and one package will be installed during the project stub creation. After that you can use Composer from the command line or through the user interface.

On this page:

- Preparing to Run Composer in the Command Line Mode
- Using Composer via the Dedicated User Interface
  - Setting up Composer in an existing project
  - Quick Composer Set-Up
  - Adding Composer Dependencies
  - Generating a Composer Project Stub

Preparing to Run Composer in the Command Line Mode

You can use the full range of Composer commands from IntelliJ IDEA only in the command line mode. To use Composer in this mode, you need to configure it as a command line tool. For more details, see Using Command Line Tools .

1. Download the Composer.phar definition file .
2. Make sure the Command Line Tool Support repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Command Line Tool Support under Tools.
4. On the Command Line Tool Support page, click the Add button. In the Choose Tool to Add dialog box that opens, choose Composer.
5. In the Composer dialog box that opens, choose the way to launch Composer:
   - Composer.phar or php script: Choose this option to launch Composer through a PHP script or have IntelliJ IDEA detect and start the launcher in the `composer.phar` archive. In this mode, IntelliJ IDEA provides coding assistance and allows to execute scripts.
     In the Path to PHP executable field, specify the location of the PHP engine installation folder. Choose the PHP installation from the list, or type the path manually, or click the Browse button … and choose the desired location in the dialog box that opens.

     In the Path to composer.phar or composer text box, specify the location of the `composer.phar` archive. Type the path manually or click the Browse button … and choose the desired location in the dialog box that opens.
   - Composer executable: choose this option to launch Composer through the `Composer` executable file. In this mode, you do not get coding assistance and cannot execute scripts because not PHP engine is appointed for it.
     In the Path to executable text box, specify the location of the `composer` executable file. Type the path manually or click the Browse button … and choose the desired location in the dialog box that opens.

   IntelliJ IDEA parses the contents of the specified `.phar` archive or executable file for Composer commands. When the file analyses is completed, IntelliJ IDEA returns to the Command Line Tools page of the Settings dialog bos where the specified file is added to the list of command line tools available in IntelliJ IDEA.

   You can have several instances of Composer configured and switch between them from one project to another by specifying the relevant one during Composer initialization .
6. In the Alias text box, specify the alias to use in calls of tool commands when running in the command line mode. Accept the default alias `c` or edit it, if necessary.
7. To activate the detected command set, select the Enable check box.
8. Customize the command set , if necessary.
9. In the Show console in area, specify where you want the Input pane for typing commands opened:
   - To have the Input pane opened in a pop-up window, choose the Pop-up option.
   - To have the Input pane opened as a text box at the bottom of the Command Line Tools Console tool window, choose the Tool window option.

For information about running the tool in the command line mode, see Running Command Line Tool Commands .

Using Composer via the Dedicated User Interface

IntelliJ IDEA provides a dedicated interface for such basic Composer operations as initializing a project and adding dependencies . In the UI mode, Composer is available at the project level. For information on using the tool from the command line, see Running Command Line Tool Commands .

Setting up Composer in an existing project

The dependencies added to a project through Composer are listed in the project-specific `composer.json` file. Therefore setting up the tool in a project starts with creating a project-specific `composer.json` file. This operation is also referred to as Composer initialization.

1. Open the project to use Composer.
2. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Composer under PHP.
3. On the Composer page that opens, specify the following:
   1. In the Path to PHP executable field, specify the location of the PHP engine installation folder. Choose the PHP installation from the list, or type the path manually, or click the Browse button … and choose the desired location in the dialog box that opens.
   2. In the Path to composer.phar text box, specify the location of the `composer.phar` archive. Type the path manually or click the Browse button … and choose the desired location in the dialog box that opens.
   3. If you have not downloaded the `composer.phar` file yet, click the Download link to have IntelliJ IDEA download `composer.phar` from the official storage and specify the folder to store the archive in. This instance of Composer will be available in the current project only. To use it in the command line mode, configure it as a command

line tool .

4. Specify how you want IntelliJ IDEA to treat Composer packages:
   - To have IntelliJ IDEA mark Composer packages added to the current project as libraries., select the Add packages as libraries check box.
     In the context of IntelliJ IDEA, any file marked as library is automatically write-protected. Library files are indexed for the only purpose which is enhancing coding assistance (i.e. code completion, syntax highlighting, navigation, and documentation lookup). Library files are not to be edited in the scope of the project where they are declared as library.

     By default, the Add packages as libraries check box is selected, which automatically blocks the possibility to edit them in the current project.
   - If you want to use Composer packages in your development and at the same time update them yourself within the scope of the same project, clear the Add packages as libraries check box.

4. Click OK to save the Composer settings and leave the Settings dialog box.
5. On the main menu, choose Tools | Composer | Init Composer. IntelliJ IDEA creates a stub of the `composer.json` file and opens it in the editor.
6. Complete the code or accept the default values. For more details, see Composer.json: Project Setup .

Quick Composer Set-Up

Alternatively, you can specify the location of the `composer.phar` archive and the PHP engine right during the Composer initialization:

1. On the main menu, choose Tools | Composer | Init Composer.
2. In the Composer Settings dialog that opens, specify the location of the `composer.phar` archive and the PHP engine to use. If you have not downloaded the `composer.phar` file yet, click the Download link to have IntelliJ IDEA download it now and specify the folder to store the file in.
3. Click OK. The dialog box is closed whereupon IntelliJ IDEA creates a stub of the `composer.json` file and opens it in the editor.
4. Complete the code or accept the default values. For more details, see Composer.json: Project Setup .

Adding Composer Dependencies

When you add a dependency to the project using Composer, a new subfolder is created under the Vendor folder, the added package is stored in this subfolder, and the added package is added to the list in the `require` section of `Composer.json` . For information on adding dependencies manually in the command line mode, see Running Command Line Tool Commands . Below are instructions for adding dependencies using the interface provided by IntelliJ IDEA.

1. Open the project where Composer is set up and choose Tools | Composer | Add Dependency on the main menu.
2. In the Add Composer Dependency dialog that opens, select the packageToAdd from the Available Packages list, possibly using the search field. The list shows all the available packages, however, the packages that have been already installed are marked with a tick.
   Choose the relevant version from the Version to install drop-down list.
3. If necessary, expand the Settings hidden area and specify the advanced installation options. In the Command line parameters text box, type the additional command line parameters to adjust the add dependency procedure. For example, to have a dependency added to the `require-dev` section instead of the default `require` section type `-dev` . For more information about Composer command line options during installation, see https://getcomposer.org/doc/03-cli.md .
4. Click Install.

When you click Finish, the `create-project` command is invoked with the selected package. This results in creating a Composer project whose configuration and structure depends on the selected package, see https://getcomposer.org/doc/03-cli.md#create-project for details. After that a IntelliJ IDEA project opens.

Generating a Composer Project Stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. The Project Category and Options dialog opens.
2. In the left-hand pane, choose PHP from the list, then choose Composer Project in the right-hand pane, and then click Next.
3. On the second page of the wizard , specify the project name and the folder where it will be created and provide the following Composer-specific information:
   1. Appoint the Composer instance to use.
      - To use commands from a previously downloaded `composer.phar` , choose Use existing composer.phar: and specify it location in the text box.
      - To have a new instance of Composer downloaded, choose Download composer.phar from getcomposer.org:. The `composer.phar` file will be saved under the project root folder specified in the Location text box.
   2. In the Package area, specify the package to install during the project creation. Select the package to add from the Available Packages list, possibly using the search field, and choose the relevant version from the Version to install drop-down list.
   3. Expand the Settings area and specify the path to the PHP interpreter to use (see Configuring Remote PHP Interpreters and Configuring Local PHP Interpreters for details). In the Command line parameters text box, type the additional command line parameters to adjust the add dependency procedure. For example, to have a package added to the `require-dev` section instead of the default `require` section type `-dev` . For more information about Composer command line options during installation, see https://getcomposer.org/doc/03-cli.md .

When you click Finish, the `create-project` command is invoked with the selected package. This results in creating a Composer project whose configuration and structure depends on the selected package, see https://getcomposer.org/doc/03-cli.md#create-project for details. After that a IntelliJ IDEA project opens.

Using Drupal with IntelliJ IDEA

You can use IntelliJ IDEA as an IDE for Drupal development including modules, themes, and core. The supported versions are 6, 7, and 8 .

IntelliJ IDEA provides integration between the Symfony2 framework and Drupal 8 while developing Drupal modules and core.

In this part:

– Preparing for Drupal Development in IntelliJ IDEA
– Drupal-Specific Coding Assistance

Preparing for Drupal Development in IntelliJ IDEA

On this page:

- Before you start
- Creating a IntelliJ IDEA project with a Drupal module
- Importing a Drupal project
- Configuring the Drupal-specific development environment
  - Configuring file associations
  - Changing the Drupal settings

Before you start

1. Download and install Drupal .
2. Make sure the Drupal plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Creating a IntelliJ IDEA project with a Drupal module

You can create a IntelliJ IDEA project that will be set up and configured in accordance with the Drupal requirements. This is done by creating a project by a Drupal Module template.

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. The Project Category and Options dialog opens.
2. In the left-hand pane, choose PHP from the list, then choose Drupal Module in the right-hand pane, and then click Next.
3. On the second page of the wizard , specify the project name and the folder where it will be created and provide the following Drupal-specific information:
   1. Specify the root folder of the Drupal installation in the Drupal installation path. Type the path manually or click the Browse button and select the relevant folder in the dialog box that opens.
   2. Select the Set up PHP | Include paths check box to have Drupal include paths automatically configured for the project. The following paths will be added to the Include Paths list on the PHP page of the Settings dialog box:
      1. `<drupal installation root>/includes`
      2. `<drupal installation root>/modules`
      3. `<drupal installation root>/sites/all/modules`

      The include paths are added to the list after you leave the dialog box.
   3. From the Version drop-down list, choose the version of Drupal to use, the supported versions are 6, 7, and 8 .

You can update integration settings at any time later in the File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.

IntelliJ IDEA generates a project stub in accordance with the selected Drupal version. For Drupal 8, a `module_name.info.yml` file is generated where information about the module is provided.

Importing a Drupal project

Besides creating a project anew according to the Drupal Module template, you can open an existing Drupal project in IntelliJ IDEA. IntelliJ IDEA recognizes the Drupal-specific structure and suggests activating the Drupal support in the project.

1. Open the folder with the required Drupal project by choosing File | Open Directory.
2. As soon as IntelliJ IDEA detects the Drupal-specific structure in the project, the following message is displayed in a pop-up window:

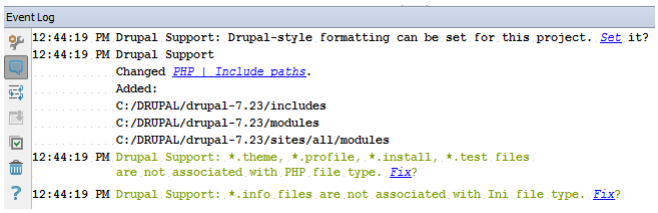
   Click the Enable link.
3. In the Drupal Module dialog box that opens, select the Enable Drupal integration check box and proceed as when creating a project with a Drupal module:
   1. Specify the root folder of the Drupal installation in the Drupal installation path.
   2. Select the Set up PHP | Include paths check box to have Drupal include paths automatically configured for the project.
   3. From the Version drop-down list, choose the version of Drupal to use, the supported versions are 6, 7, and 8 .

You can update integration settings at any time later in the File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.

Configuring the Drupal-specific development environment

Whether you enable the Drupal support in an existing IntelliJ IDEA project or create a new project with a Drupal module, IntelliJ IDEA checks if the development environment is configured properly for Drupal development.

Any detected inconsistency is reported in the Event Log tool window and as a pop-up. For each discrepancy IntelliJ IDEA suggests a fix. To apply a suggestion, click the link next to the reported event.



Configuring file associations

To enable code highlighting, you need to set some additional file type associations to make IntelliJ IDEA recognize and treat them as `php` files. Do one of the following:

- In the Event Log tool window, click the Fix link next to the `Drupal support: <*.file extension> files are not associated with PHP file type` message.
- Use the File Types page of the Settings dialog box:
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click File Types under Editor.

2. In the Registered Patterns area, add the patterns that defined Drupal files to be recognized as `php` files. To do that, click the Add button + and type the pattern in the Add Wildcard dialog box that opens.

Changing the Drupal settings

No matter whether you enable the Drupal support in an existing IntelliJ IDEA project or create a new Drupal module, you can update the Drupal settings at any time on the Drupal page of the Settings / Preferences Dialog .

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Drupal under Languages&Frameworks.
2. Click Drupal and make the necessary changes on the Drupal page that opens:
   - To turn off the Drupal integration, clear the Enable Drupal integration check box. Note that this automatically excludes the folders specified as include paths.
   - Change the Drupal installation directory and the version of Drupal to use.
3. To change the include paths, click PHP and make the required changes in the Include Paths area of the PHP page that opens. Note that is the Drupal integration for is disabled, the paths are automatically excluded.
4. To change the code style settings, click Code Style, and then click PHP. On the Code Style: PHP page that opens, update the settings as necessary.

Drupal-Specific Coding Assistance

On this page:

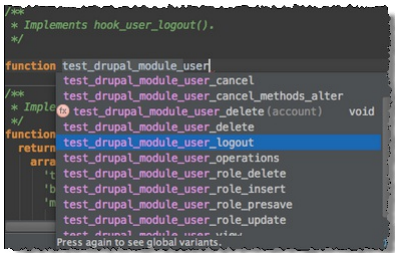
- Using Drupal hooks in IntelliJ IDEA
  - Completing hook declarations
  - Navigating to a hook invocation from the editor
  - Quick Documentation look-up for hooks
- Setting the Drupal code style in a project
- Checking code against the Drupal coding standards
- Viewing Drupal API documentation from IntelliJ IDEA
- Using the Drush command line tool
- Integration between Drupal 8 and Symfony2
  - Coding Assistance for Drupal8 - Symfony2 Integration

Using Drupal hooks in IntelliJ IDEA

IntelliJ IDEA provides full native support of Drupal hooks in `.module` files.

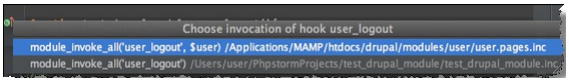
Completing hook declarations

IntelliJ IDEA indexes any hook invocation whereupon hook names become available in code completion for creating hook implementations. Start typing a hook name and press `Ctrl+Space`.



Navigating to a hook invocation from the editor

- To navigate to a hook invocation from the editor, click the icon in the gutter.
- In case of multiple invocations, IntelliJ IDEA displays a list of available hook invocations so you can choose which one to navigate to.



You will be navigated to the line where the relevant hook is invoked with `module_invoke_all()`, `module_invoke()`, or `drupal_alter()`.

Quick Documentation look-up for hooks

You can view quick documentation for a Drupal hook in IntelliJ IDEA. Documentation is taken from `.api.php` files provided by `core` and many other modules for reference purposes.

To view documentation on a hook, position the cursor at the hook name in question and choose View | Quick Documentation Lookup or press `Ctrl+Q`.

Setting the Drupal code style in a project

IntelliJ IDEA automatically offers to apply the pre-configured Drupal Coding Standards (code style) if a project is recognized as a Drupal Module, or if the Drupal integration is enabled in an existing project, or when you create a new project with a a Drupal Module. However, you can at any time change or customize this setting on the Code Style: PHP page of the Settings dialog box.

To have the pre-configured Drupal code style applied to a project, do one of the following:

- In the Event Log tool window, click the Set link next to the `Drupal-style formatting can be set for this project` message.
- Use the Code Style: PHP page of the Settings dialog box:
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click PHP under Code Style.
  2. On the Code Style: PHP page that opens, click the Set from link.
  3. On the pop-up menu, choose Predefined, then choose Drupal.

Checking code against the Drupal coding standards

With IntelliJ IDEA, you can use two tools that detect violations against the Drupal coding standard: Coder and PHP Code Sniffer. They ensure your code remains clean and consistent and help prevent some common semantic errors made by developers.

- To use PHP Code Sniffer:
  1. Install PHP Code Sniffer as a PEAR package or using the Composer tool. For detailed installation instructions, see https://github.com/squizlabs/PHP_CodeSniffer.
  2. Register PHP Code Sniffer with IntelliJ IDEA and configure it as a IntelliJ IDEA inspection. For details, see Using PHP Code Sniffer Tool.
- To use Coder, the native Drupal code quality tool:
  1. Download the Drupal Coder module (7x-2.0 version is recommended). You don't need to install or use the Drupal Coder module, it contains the Drupal Coding Standards inside.
  2. Unpack the downloaded archive and find the `coder_sniffer/Drupal` subdirectory inside. You need to move the contents of this directory to `<php installation folder>/CodeSniffer/Standards/Drupal`. This directory must contain the `ruleset.xml` file and some other subdirectories.

Viewing Drupal API documentation from IntelliJ IDEA

To reach the Drupal API Documentation at https://api.drupal.org/api/drupal right from IntelliJ IDEA, select the symbol you are interested in and choose Search in Drupal API on the context menu of the selection.

Using the Drush command line tool

Drush is a command line shell and scripting interface for Drupal. With IntelliJ IDEA, you can use Drush 5.8 and higher.

1. Download and install Drush as described at https://github.com/drush-ops/drush .
2. Integrate Drush with IntelliJ IDEA by configuring it a s Command Line Tool:
   1. Open the Settings dialog box and click Command Line Tool Support.
   2. On the Command Line Tool Support page, click the Add button. In the Choose Tool to Add dialog box that opens, choose Drush and specify the tool's visibility. The available options are Project (available only in the current project) and Global (available across all IntelliJ IDEA projects). Click OK.
   3. In the Drush dialog bo that opes, specify the path to the Drush executable file. IntelliJ IDEA automatically fills in the default location, which is usually `C:/ProgramData/Drush/drush.bat` on Windows and `/usr/bin/drush` on Mac OS or Linux. If you followed the standard installation procedure, the predefined path will be correct, just click OK, whereupon IntelliJ IDEA loads command definitions automatically and returns to the Command Line Tool Support page.
   4. In the Alias text box, specify the alias to use in calls of tool commands. Accept the default alias or edit it, if necessary.
   5. To activate the detected command set, select the Enable check box.
3. To run a Drush command, choose Tools | Run Command... on the main menu or press `Ctrl+Shift+X` (`Cmd-Shift-X` on OS X).Ctrl+Shift+X (Cmd-Shift-X on OS X).

For details about configuring and running command line tools in IntelliJ IDEA, see Using Command Line Tools .

Integration between Drupal 8 and Symfony2

IntelliJ IDEA provides close integration between Drupal, version 8 and Symfony2 which connects Symfony2 components and Drupal infrastructure. To take advantage from this integration:

1. Install and activate the Drupal Symfony2 Bridge plugin on the Plugins page of the Settings dialog box as described in Installing, Updating and Uninstalling Repository Plugins and in Enabling and Disabling Plugins .
   The Drupal Symfony2 Bridge plugin depends on the Symfony2, which will be installed automatically.
2. To get advanced annotations support, install the PHP Annotations plugin as described in Installing, Updating and Uninstalling Repository Plugins and in Enabling and Disabling Plugins .

Coding Assistance for Drupal8 - Symfony2 Integration

- Completion for major Drupal-specific parameters in `<module_name>.info.yml` files that contain information about Drupal modules.
- Strings suitable for use inside Drupal-specific t() function are indexed across your project and offered for completion.
- Completion for relevant `.yml` key values in url() and other Drupal API functions, which makes search for the right value easier.
- Navigation to the `.yml` file by pressing `Ctrl+B` or choosing Navigate | Go To Declaration.
- Full support for service containers described in `.yml` files, including completion and navigation with `Ctrl+B` .
- Support of the Twig template engine , which is now the default template engine for Drupal 8, including completion, navigation, and recognition of Drupal functions. See also Twig in Drupal 8 .

Using Blade Templates

IntelliJ IDEA provides full support of the Laravel Blade template engine up to version 5.1. This support involves:

- Highlighting of Blade syntax in template files.
- Code completion for all Blade directives, both predefined and custom, as well as for braces.
- In `@for` and `@foreach` directives, variable introduction is offered with autocompletion inside the code constructs.
- Expanding and folding sections defined through the `@section` directive. A block of code between a `@section` directive and a closing directive (for example, `@stop`) can be expanded or folded by clicking or in the gutter area.
- Dedicated Blade-specific code inspections, for example an inspection to check that a section opened with `@section` directive is closed with one of the corresponding directives.
- Blade-aware navigation with `Ctrl+B` includes links to templates in `@extends` and `@include` directives.
- Finding usages (`Alt+F7`) can be invoked on a file name or a symbol in the code to show all the usages of a template across your codebase. Currently this functionality is available only inside other templates, but not from views.
- Customizing predefined Blade directives and defining custom directives for Blade templates.

On this page:

- Preparing to Use Blade Templates
- Adding, Editing, and Removing Blade Directives
- Configuring Blade Delimiters

Preparing to Use Blade Templates

Before you start, make sure the PHP and Blade plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Adding, Editing, and Removing Blade Directives

In IntelliJ IDEA, Blade directives are managed in the Directives tab of the Blade Page. The tab lists all the currently available Blade directives, for those that have parameters, the prefixes and suffixes are also shown. When you start, the list contains only predefined directives. You can edit these directives as well as create custom ones.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Blade under PHP. On the Blade page that opens, switch to the Directives tab. The tab shows a list of all currently available directives.
2. Do one of the following:
   - To define a new directive, click the Add directive button and then specify the directive's name in the Name text box. If the new directives requires a prefix and a suffix, select the Has parameter check box and type the prefix and suffix to use in the Prefix and Suffix text boxes respectively. IntelliJ IDEA will automatically enclose the prefix and suffix in opening and closing brackets and quotes and add a colon separator `:` so the parameters will look as follows: `("<prefix>:<suffix>")` .
   - To edit an existing directive, select it in the list and change the values in the text boxes below. To restore the original definition, click the Reset to defaults button .
   - To remove a directive from the list, select it and click the Remove directive button .

Configuring Blade Delimiters

IntelliJ IDEA recognizes Blade templates and provides error highlighting and code completion for them based on the delimiters you specify. These delimiters are managed in the Text Tags tab of the Blade Page .

The fields in the tab show the opening and closing characters for raw tags , content tags , and escaped tags .

The fields are filled in with the default values in compliance with Blade Templates 5.1 . If you are using an earlier version, you can specify the relevant custom delimiters and IntelliJ IDEA will provide coding assistance according to the new rules.

Using WordPress Content Management System

WordPress WordPress WordPress WordPress

You can run WordPress from IntelliJ IDEA in two modes:

- Through a dedicated user interface. To get access to this functionality in a project, you need to download WordPress, register it in IntelliJ IDEA, and activated the WordPress integration within the current project.
- Through a set of command line tools. In this case, you need to download WP-CLI and configure it as a IntelliJ IDEA command line tools, see Creating a Command Line Tool .

In this section:

- Preparing to Use WordPress
- WordPress Specific Coding Assistance
- Using the WordPress Command Line Tool WP-CLI

Preparing to Use WordPress

IntelliJ IDEA provides a dedicated interface for developing and running WordPress applications and provides WordPress-aware coding assistance, see WordPress Specific Coding Assistance . To get access to this functionality in a project, you need to download WordPress, register it in IntelliJ IDEA, and activated the WordPress integration within the current project.

Alternatively, you can download WordPress and create a stub of a WordPress-targeted project. In this case, the WordPress integration in the project will be activated automatically.

Whether you enable the WordPress integration in an existing project or create a stub of a WordPress application, IntelliJ IDEA checks if the development environment is configured properly for WordPress development. If the configuration does not meet the requirements, IntelliJ IDEA displays a pop-up window with a Fix link.

On this page:

- Before You Start
- Downloading and Installing WordPress
- Activating the WordPress Installation in a Project
- Generating a WordPress application stub

Before You Start

1. Make sure the PHP interpreter is configured in IntelliJ IDEA on the PHP page , as described in Enabling PHP Support .
2. Make sure the PHP and WordPress Support plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.
3. Configure the PHP interpreter to use WordPress with, as described Configuring Remote PHP Interpreters . Note that WordPress requires PHP 5.3 or higher.

Downloading and Installing WordPress

1. Download the `WordPress.zip` archive at https://wordpress.org/download/ .
2. Extract the archive:
   - To have the possibility to run your WordPress application on a local Web server, store the extracted file to the document root of the server where the PHP interpreter is configured. For local development environment with the Apache HTTPD Web server, extract WordPress.zip to the `htdocs` folder.
   - To run your WordPress application on an in-place server, store the extracted file under the project root.
   - To run your WordPress application on a remote server, store the extracted files in your project and then configure automatic upload of them to the document root of the remote server.
     Learn more about server access configurations, see Configuring Synchronization with a Web Server . For information on configuring upload to the server, see Uploading and Downloading Files .
   - If you are not going to run your WordPress application but just need to get coding assistance from IntelliJ IDEA, store the extracted files anywhere on your computer. In this case, you will have to configure the installation as an include path, see Configuring Include Paths .

To run WordPress in the command line mode, you will need a set of command line tools which you can acquire by installing the `wp-cli/wp-cli` package using the Composer dependency manager or by downloading the `wp-cli.phar` archive. For details, see Using the WordPress Command Line Tool WP-CLI .

Activating the WordPress Installation in a Project

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click WordPress under Tools.
2. On the WordPress page that opens, select the Enable WordPress Integration check box.
3. In the WordPress Installation Path text box, specify the folder where WordPress is installed. This folder should contain the `wp-admin` and `wp-includes` subdirectories.
4. Click Apply to save the WordPress registration and click PHP.
5. On the PHP page that opens, add the path to the WordPress installation folder to the Include Paths list: click the + button and specify the path to the installation folder in the dialog box that opens. Learn more in Configuring Include Paths .

Generating a WordPress application stub

During project creation, IntelliJ IDEA can generate a project stub for developing WordPress applications. The project structure is set up in accordance with the WordPress requirements.

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen. The Project Category and Options dialog opens.
2. In the left-hand pane, choose PHP from the list, then choose WordPress Plugin in the right-hand pane, and then click Next.
3. On the second page of the wizard, specify the project name and the folder where it will be created. In the WordPress Installation Path text box, specify the folder where WordPress is installed. This folder should contain the `wp-admin` and `wp-includes` subdirectories. Click Finish to start the project stub generation.
4. If the newly created project is outside the WordPress installation folder, configure it as an external library by adding it to the list of included path.
   1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
   2. On the PHP page that opens, add the path to the WordPress installation folder to the Include Paths list: click the + button and specify the path to the installation folder in the dialog box that opens. Learn more in Configuring Include Paths .

WordPress Specific Coding Assistance

IntelliJ IDEA provides full coding assistance for developing WordPress including WordPress-aware code completion, search for hook registration functions and functions specified as hook registration parameters, navigation between hook registrations and the hook invocations, callbacks, the possibility to configure the coding style in accordance with the WordPress code style, viewing the official WordPress documentation from the IntelliJ IDEA editor, etc.

On this page:

- Adding the WordPress to the project
  - Configuring the WordPress installation as a project include path
  - Adding the wp-content folder to the project
- Configuring WordPress Code Style
- WordPress Hooks Support
  - Completion for WordPress Action and Filter functions parameters
  - Navigation from a Hook Registration to the Hook Invocation
  - Callbacks from Hook Registration
  - Navigating to Hook Invocations
  - Searching for Hook Registration Functions
- Viewing Official WordPress Documentation from IntelliJ IDEA

Adding the WordPress to the project

To take advantage of coding assistance provided by IntelliJ IDEA, the WordPress installation you are working with should be configured in the project as an external library. As a result, the WordPress core file will be involved in indexing which is the basis for resolving references and providing code completion, navigation, search. etc.

If the `wp-content` folder is outside the WordPress installation, it has to be added either as an include path or as a content root.

Configuring the WordPress installation as a project include path

To add WordPress to your project as an external library, you need to add it to the list of include paths. Learn more in Configuring Include Paths .

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
2. On the PHP page that opens, add the path to the WordPress installation folder to the Include Paths list: click the + button and specify the path to the installation folder in the dialog box that opens.

Adding the wp-content folder to the project

If the `wp-content` folder is located outside the WordPress installation, you need to add it to the project individually, apart from the WordPress core files. This can be done in two ways:

- To have the `wp-content` folder involved in indexing without putting it under the project version control, add it to the list of include paths:
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click PHP under Languages&Frameworks.
  2. On the PHP page that opens, add the path to the `wp-content` folder to the Include Paths list: click the + button and specify the path to the folder in the dialog box that opens.

- To have the `wp-content` folder involved in indexing and put it under the project version control, add it as a content root:
  1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Directories under Project:<project name>.
  2. On the Directories page that opens, click the Add Content Root button . In the dialog that opens , locate the `wp-content` directory and click OK.

  Learn more about configuring content roots in Configuring Content Roots . For information about adding files and folders to version control, see Enabling Version Control .

Configuring WordPress Code Style

In IntelliJ IDEA, you can use the WordPress Code Style in accordance with the WordPress coding standards . To configure the native WordPress code style, do one of the following:

- 1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Editor node, and then click PHP under Code Style.
  2. On the Code Style. PHP page that opens, click the Set from... link.
  3. On the context menu, choose Predefined Style, then choose WordPress.

- Upon activating of the WordPress integration, IntelliJ IDEA displays a pop-up window offering you to set the WordPress code style. Click the Set it link in the pop-up window and set the code style on the Code Style. PHP page that opens.

Learn more in Configuring Code Style .

WordPress Hooks Support

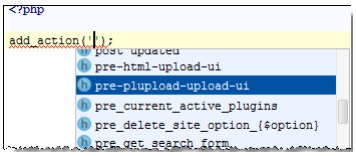
IntelliJ IDEA indexes all hooks declared in the WordPress core and in the included plugins. This makes the basis for coding assistance when working with hooks.

Completion for WordPress Action and Filter functions parameters

Hook names are available in code completion of standard parameters for action and filter functions (`add_action()` and `add_filter()`). To invoke completion for a parameter:

1. Start typing the declaration of an `add_action()` or `add_filter()` function.

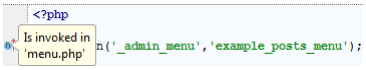
2. Press `Ctrl+Space` and choose the relevant parameter from the list.



Navigation from a Hook Registration to the Hook Invocation

To navigate from a hook registration (`add_action()` or `add_filter()` function) to the hook invocation:

– Click the icon in the gutter area next to the hook registration to navigate from.



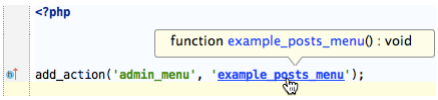
IntelliJ IDEA opens the file where the hook is invoked and positions the cursor at the invocation method, in the current example it is `do_action`.

>

Callbacks from Hook Registration

You can navigate to the declaration of the function or method specified as the second parameter of a hook registration (`add_action()` or `add_filter()` function). To do that:

– With the `Ctrl` keyboard key pressed, hover the cursor over the parameter of interest. IntelliJ IDEA displays a pop-up information message with the definition of the function or method specified as this parameter:



The parameter turns into a link. When you click this link, IntelliJ IDEA opens the file where the function or method was declared and positions the cursor and declaration.

>

Navigating to Hook Invocations

To search for a hook invocation and navigate to it, do one of the following:

– Use the Navigate to Symbol functionality:

1. Choose Navigate | Symbol on the main menu.
2. In the Enter symbol name pop-up window that appears, start typing the hook name in the search field. If necessary, select the Include non-project items check box. The contents of the list below the search field change as you type.
3. Click the relevant hook from the list. %product" opens the file where it is invoked and positions the cursor at the invocation.

– Use the Search Everywhere functionality:

1. Press the Shift keyboard key twice.
2. start typing the hook name in the search field. If necessary, select the Include non-project items check box. The contents of the list below the search field change as you type.
3. Click the relevant hook from the list. %product" opens the file where it is invoked and positions the cursor at the invocation.

Learn more about search and navigation in Searching Through the Source Code and Navigating Through the Source Code .

Searching for Hook Registration Functions

You can search for occurrences of a hook registration function and for occurrences of a function or method specified as the second parameter of a hook registration.

– To get a list of usages for a hook registration function:

1. Position the cursor at the name of the desired hook registration function and press `Alt+F7` or choose Edit | Find | Find Usages on the main menu.
2. From the pop-up list that shows the available options in terms of hook/function name, choose the relevant option. The Find tool window opens with a new tab showing all the detected occurrences of the selected function.
3. Explore the listed occurrences. To navigate to the relevant one, click it. IntelliJ IDEA opens the file with the selected function or method and positions the cursor at it.

– To get a list of occurrences of a function or method specified as the second parameter of a hook registration:

1. Position the cursor at the parameter of interest and press `Alt+F7` or choose Edit | Find | Find Usages on the main menu. The Find tool window opens with a new tab showing all the detected occurrences of the function or method specified as the parameter on which the search was invoked.
2. Explore the listed occurrences. To navigate to the relevant one, click it. IntelliJ IDEA opens the file with the selected function or method and positions the cursor at it.

Learn more in Finding Usages .

Viewing Official WordPress Documentation from IntelliJ IDEA

You can view the official WordPress Documentation at http://wordpress.org/ right from the IntelliJ IDEA editor.

Select the text you are interested in

1. Select the text you are interested in.
2. On the context menu of the selection, choose Search on WordPress.org.
   IntelliJ IDEA opens the page with the relevant documentation in the default browser.

Using the WordPress Command Line Tool WP-CLI

To run WordPress in the command line mode, you will need a set of command line tools which you can acquire by installing the `wp-cli/wp-cli` package using the Composer dependency manager or by downloading the `wp-cli.phar` archive. The downloaded command line tool must be registered in IntelliJ IDEA as described in Using Command Line Tools .

For information about running the tool in the command line mode, see Running Command Line Tool Commands .

On this page:

– Before you start
– Installing the wp-cli package using the Composer dependency manager
– Downloading the wp-cli.phar archive
– Configuring wp-cli as a IntelliJ IDEA command line tool
– Running the wp-cli tool

Before you start

1. Make sure the PHP interpreter is configured in IntelliJ IDEA on the PHP page , as described in Enabling PHP Support .
2. Make sure the PHP and Command Line Tool Support plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.
3. Configure the PHP interpreter to use WordPress with, as described Configuring Remote PHP Interpreters . Note that WordPress requires PHP 5.3 or higher.

Installing the wp-cli package using the Composer dependency manager

Before you start, make sure Composer is installed on your machine and initialized in the current project, see Using Composer Dependency Manager .

1. On the main menu, choose Tools | Composer | Add Dependency.
2. In the Add Composer Dependency dialog that opens, select the `wp-cli/wp-cli` package from the Available Packages list, possibly using the search field. The list shows all the available packages, however, the packages that have been already installed are marked with a tick.
   Choose the relevant version from the Version to install drop-down list.
3. If necessary, expand the Settings hidden area and specify the advanced installation options. In the Command line parameters text box, type the additional command line parameters to adjust the add dependency procedure. For example, to have the package added to the `require-dev` section instead of the default `require` section type `-dev` . For more information about Composer command line options during installation, see https://getcomposer.org/doc/03-cli.md .
4. Click Install.

When you click Finish, the `create-project` command is invoked with the selected package. This results in creating a Composer project whose configuration and structure depends on the selected package, see https://getcomposer.org/doc/03-cli.md#create-project for details. After that a IntelliJ IDEA project opens.

Downloading the wp-cli.phar archive

– Download the `wp-cli.phar` at http://wp-cli.org/ .

Configuring wp-cli as a IntelliJ IDEA command line tool

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Command Line Tool Support under Tools.
2. On the Command Line Tool Support page, click the Add button. In the Choose Tool to Add dialog box that opens, choose WP-CLI.
3. In the WP-CLI dialog box that opens, choose the way to run WordPress:
   – Installed via PHAR: Choose this option to launch WordPress through a PHP script or have IntelliJ IDEA detect and start the launcher in the `wp-cli.phar` archive.
   In the Path to PHP executable field, specify the location of the PHP engine installation folder. Choose the PHP installation from the list, or type the path manually, or click the Browse button and choose the desired location in the dialog box that opens.

   In the Path to phar text box, specify the location of the `wp-cli.phar` archive. Type the path manually or click the Browse button and choose the desired location in the dialog box that opens.
   – Executable available (installed via Composer, etc.): choose this option to launch WordPress through an executable file which is available when you install WordPress using a package management tool, for example, Composer.
   In the Path to wp.bat field, specify the location of the `wp.bat` or `wp` executable file. If you used Composer, the default location is `\vendor\wp\cli\bin\wp` or `\vendor\wp\cli\bin\wp.bat` . Type the path manually or click the Browse button and choose the desired location in the dialog box that opens.
4. When you click OK, the WP-CLI dialog box closes and IntelliJ IDEA brings you back to the Command Line Tool Support page, where wp is added to the list of available tools.
5. From the Console encoding drop-down list, choose the character set to show the tool's output in the Command Line Tools Console Tool Window .
6. In the Show console in area, specify where you want the Input pane for typing commands opened:
   – To have the Input pane opened in a pop-up window, choose the Pop-up option.
   – To have the Input pane opened as a text box at the bottom of the Command Line Tools Console tool window, choose the Tool window option.

Running the wp-cli tool

You can run WP-CLI commands and analyze their output right in IntelliJ IDEA using the dedicated Command Line Tools Console tool window, just as the commands of any other command line tools, see Running Command Line Tool Commands .

Using Google App Engine for PHP

With IntelliJ IDEA, you can develop a PHP application for running in the Google PHP Runtime Environment . IntelliJ IDEA supports all the major Google App Engine for PHP development practices and provides the possibility to run and debug your applications locally before uploading them to the runtime environment.

In this section:

- Preparing to Develop a Google App for PHP Application
- Running, Debugging, and Uploading an Application to Google App Engine for PHP

Google App Engine support in IntelliJ IDEA

- A dedicated project type with a specific directory structure and configuration file.
- Generating and analyzing a model dependency diagram for the Google App Engine models.
- A dedicated App Engine for PHP run/debug configuration for running and debugging applications on the PHP Development Server which comes bundled with the Google App Engine for PHP SDK.
- Uploading applications using the command of the Tools menu.

Preparing to Develop a Google App for PHP Application

IntelliJ IDEA takes care of creating the specific directory structure and settings. IntelliJ IDEA can either generate a Google App Engine for PHP-specific project stub or you can enable support of Google App Engine for PHP in an existing project.

- Before You Start
- Creating a Google App Engine for PHP Application Project Stub
- Activating the Google App Engine Support in an Existing Project

Before You Start

1. Download and install Python , version 2.7.
2. Download and install the Google App Engine for PHP .
3. Install and enable the Google App Engine Support for PHP plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
4. Create a Google account at http://www.google.com .
5. Create an application on the Create application page of the Google App Engine and remember its application ID, you will have to specify it during the creation of a project.

Creating a Google App Engine for PHP Application Project Stub

1. Choose File | New | Project on the main menu or click the New Project button on the Welcome screen.
2. In the Project Category and Options dialog, which is the first page of the New Project wizard, choose PHP in the left-hand pane.
3. In the right-hand pane, choose App Engine Project and click Next.
4. On the second page of the wizard, specify the following:
   1. The name of the project and the folder to create it in.
   2. The version of the template to use.
   3. In the Application ID text box, type the identifier of your application as you specified it on the Create application page.
   4. In the SDK directory test box,
   5. The location of the Python executable file. Type the path manually in the Python executable text box or click the Browse button and choose the Python executable in the dialog box that opens.

   Click Finish when ready.

IntelliJ IDEA creates a project with the `main.php` file and the app.yaml configuration file. The `app.yaml` file contains the generated settings for the runtime (should be `php55`), the api_version , the applicationID , the application version (by default is 1), the threadsafe element with the value `true` to enable sending multiple, parallel requests, and the Script handlers `url` and `script` (with the value `main.php`).

Activating the Google App Engine Support in an Existing Project

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X. Expand the Languages&Frameworks node, and then click Google App Engine for PHP under PHP.
2. On the Google App Engine for PHP page that opens, select the Enable Google App Engine for PHP support check box and specify the following:
   1. The folder where the Google App Engine for PHP SDK is stored.
   2. The path to the Python 2.7 executable file.
   3. In the App Engine Account Settings area, choose the way to authenticate to the development server, the available options are:
      - Use passwordless login via OAuth2: choose this option to use the OAuth 2.0 protocol . To save the token achieved through the Google Developers Console , clear the Do not save token check box.
      - Log in with email and password: choose this option to use your Gmail address and password.

Running, Debugging, and Uploading an Application to Google App Engine for PHP

IntelliJ IDEA provides the possibility to run and debug your applications locally on the PHP Development Server before uploading them to the Google PHP Runtime Environment. Applications are launched locally according to a dedicated App Engine for PHP run/debug configuration.

On this page:

- Working with the app.yaml file
- Creating an App Engine for PHP Run Configuration
- Running an Application on the PHP Development Server
- Debugging an Application on the PHP Development Server
- Uploading an Application to the Google PHP Runtime Environment

Working with the app.yaml file

A Google App Engine PHP application is configured in the app.yaml file. If you generate a project stub (see Preparing to Develop a Google App for PHP Application), the `app.yaml` file is created automatically. If you activate Google App Engine support in an existing project, you need to create an `app.yaml` file manually. In either case, make sure the `runtime` property is set to `php55`, because the Google App Engine for PHP SDK contains a built-in PHP 5.5 interpreter.

Creating an App Engine for PHP Run Configuration

1. On the main menu, choose Run | Edit Configurations.
2. Click the Add New Configuration toolbar button, and choose App Engine for PHP on the context menu.
3. In the Run/Debug Configuration: App Engine for PHP dialog box that opens, specify the host to run the development server and the application on, the default is `localhost`, and the port through which IntelliJ IDEA will communicate with the development server, the default port is `8080`.
4. Optionally in the Commnad Line area, specify the settings for running and debugging your application on the PHP development server in the command-line mode:
   - In the Interpreter options field, specify the options to be passed to the PHP executable file of the built-in PHP interpreter, see Command-Line Arguments for details.
   - In the Custom working directory field, specify the location of the files that are outside the folder with your sources and are referenced through relative paths. Type the path manually or click the Browse button and select the desired folder in the dialog that opens.
   - In the Environment variables field, In this field, specify the environment variables be passed to the built-in server. See Environment Variables in Apache for details.

Running an Application on the PHP Development Server

1. Create an App Engine for PHP run configuration as described above.
2. Choose the configuration from the list and click on the toolbar.
3. View and analyze the output of the application in console of the Run tool window.
4. To view the application execution results, open your browser at `http://localhost:8080`.

Debugging an Application on the PHP Development Server

1. Create an App Engine for PHP run configuration as described above.
2. Set the breakpoints in your code, see Using Breakpoints.
3. Choose the configuration from the list and click
4. In the Debug tool window that opens, step through the program, stop and resume the program, examine it when suspended, etc. See Stepping Through the Program, Pausing and Resuming the Debugger Session, and Examining Suspended Program for details.,

Uploading an Application to the Google PHP Runtime Environment

After you have tested your application by running and debugging it locally on the development server, you can deploy it to the Google PHP Runtime Environment.

1. On the main menu, choose Tools | Google App Engine for PHP | Upload App Engine PHP app.
2. Visit the application at `http://<your-application-id>.appspot.com/`.

Play Framework 1.x

IntelliJ IDEA supports the version 1.x.x of the Play framework . The necessary integration with the framework is provided by the Playframework Support plugin bundled with with the IDE.

A Play application in IntelliJ IDEA corresponds to a Java module .

> **Note** If you want to use IntelliJ IDEAjust for running the `play` command-line utility (also known as the Play console), you can install and use the version 2.x.x of the Play framework.

On this page:

- Play framework support overview
- Preparing for Play application development
- Creating a Play application
- Accessing the play command-line utility in IntelliJ IDEA
- Adding Play modules to an IntelliJ IDEA module
- Running a Play application
- Running Play application tests
- Debugging a Play application: process overview
- Creating a run/debug configuration for debugging
- Modifying the run configuration for running the application

Play framework support overview

Please note that the Play framework version 1.x is supported by the IntelliJ IDEA version 11. The later versions of IntelliJ IDEA support the Play 2.x version. Moreover, for the IntelliJ IDEA version 14.0, the Play 2.x framework is merged with the Scala plugin.

The Play framework support in IntelliJ IDEA includes:

- An ability to use the `play` command-line utility (the Play console) right in the IDE.
- Coding assistance:
  - Navigation between the methods of `Application.java` and the corresponding templates.
  - Updating templates through the variables of the corresponding `Application.java` methods.
  - Code completion for keywords, labels, variables, parameters and functions.
  - Error and syntax highlighting.
  - Code formatting and folding .
- Advanced source code search and replace capabilities .
- Structure view .

Preparing for Play application development

To prepare for Play application development:

1. Download and install the desired Play framework version supported by IntelliJ IDEA . See the installation instructions .
2. Create a new Play application (`play new`) or get an existing one (the one that you are going to work with in IntelliJ IDEA).
3. Prepare your application for opening it in IntelliJ IDEA (`play idea` or `play idealize`).
4. Make sure that the Play framework settings are properly specified in IntelliJ IDEA.
5. Open the application in IntelliJ IDEA.
6. Optionally, convert the project into the directory-based format . See Converting Projects Into Directory-Based Format .

Creating a Play application

To create a new Play application, run the `play new` command in a command-line shell, outside IntelliJ IDEA:

1. Open your command-line shell.
2. Switch to the directory in which you want to create your Play application.
3. Run the following command:

```
play new <dir>
```

   where `<dir>` is the name of your new Play application directory (e.g. `helloworld`) which will be created in the current directory.
4. When asked for the application name, type the name and press `Enter` .

As a result, the Play application with the specified name is created in the specified directory.

Preparing a Play application for opening it in IntelliJ IDEA

To prepare a Play application for opening it in IntelliJ IDEA, run the `play idea` or the `play idealize` command from the root directory of your Play application:

1. Open your command-line shell.
2. Switch to the root directory of your Play application.
3. Run the following command:

```
play idea
```

(Alternatively, you can use the `play idealize` command.)

As a result, all the necessary IntelliJ IDEA configuration files are created in the current directory. These include the `<app_name>.ipr` project file which you can now open in IntelliJ IDEA .

Specifying Play framework settings

In IntelliJ IDEA, the Play framework settings are specified in the Settings dialog, on the Play Configuration page.

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane of the dialog, select Play Configuration.
3. On the Play Configuration page that opens in the right-hand part of the dialog:
   - In the Home field, specify the Play framework installation directory.
   - In the Working directory field, specify the Play framework working directory. This is the directory from which the commands of the `play` command-line utility are run.
   - Depending on your preferences, turn the Show on console run option on or off.
4. Click OK in the Settings dialog.

Opening a Play application in IntelliJ IDEA

To open a Play application in IntelliJ IDEA, you should open the `<app_name>.ipr` project file you have previously generated :

1. Select File | Open.
2. In the Open File or Project dialog, go to your Play application root folder, select the `<app_name>.ipr` file, and click OK.

In the project that opens your Play application is represented by an IntelliJ IDEA module .

If necessary, convert your project into the directory-based format . See Converting Projects Into Directory-Based Format .

An alternative way to create an IntelliJ IDEA project for a Play application

The quickest and the most convenient way to start working with a Play application in IntelliJ IDEA is to run the play idea command and then open the generated .ipr file in IntelliJ IDEA.

As an alternative, you can create a new project for your Play application sources using File | New | Project from Existing Sources and then add the necessary Play framework assets (`<play_dir>\framework\lib` and `<play_dir>\framework\play-<version>.jar`) to dependencies of the resulting IntelliJ IDEA module:

1. Select File | New | Project from Existing Sources.
2. In the dialog that opens , select your Play application root directory.
3. On the first page of the Import Project wizard, select Create project from existing sources and click Next.
4. On the next page of the wizard , in the Project location field, specify the path to your Play application root directory. The rest of the settings are not that important. Click Next.
5. Follow the instructions of the wizard. (Normally, all you have to do is to click Next on each of the pages accepting the default settings.)
   When the project has been created, add the necessary module dependencies:
6. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
7. In the left-hand pane of the dialog, select Modules.
8. On the Module page shown in the right-hand part of the dialog, select the Dependencies tab , click + and select Jars or directories.
9. In the Attach Files or Directories dialog, go to the Play framework installation directory, select the directory `framework\lib` and the file `framework\play-<version>.jar` , and click OK.
0. Click OK in the Project Structure dialog.

Accessing the play command-line utility in IntelliJ IDEA

If a Play application is currently open in IntelliJ IDEA, you can access the `play` command-line utility (the Play console) and run it right from the IDE:

1. Select Tools | Play with Playframework.
2. In the Play Configuration dialog specify the Play framework settings and click OK. (This dialog is not shown if the Show on console run option is off in the Play framework settings .)

As a result, the `play` command-line utility is started in the Run tool window .

Adding Play modules to an IntelliJ IDEA module

The Play modules used by your Play application should be added to the corresponding IntelliJ IDEA module as the module content roots . For example, if your application uses (or is about to be using) the Play module `secure` :

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. In the left-hand pane of the dialog, select Modules.
3. On the Module page shown in the right-hand part of the dialog, select the Sources tab and click Add Content Root.
4. In the Select content root directory dialog, go to the Play framework installation folder, select the directory `modules\secure` , and click OK. (If the Play module you want isn't included in the Play framework distribution, you should download that module first to make it available locally.)
5. Click OK in the Project Structure dialog.

Running a Play application

To run your play application, you can use the `play run` command:

1. Start the play command-line utility (Tools | Play with Playframework).
2. In the Run tool window, after `play` , type `run` and press `Enter` .
3. Open a Web browser and go to `http://localhost:9000` to see the application home page.

As an alternative, you can create an IntelliJ IDEA run/debug configuration , and use that configuration for running your application.

To create a run configuration for running your Play application:

1. Open the Run/Debug Configurations dialog (e.g. Run | Edit Configurations).
2. Click + (`Alt+Insert`) and select Application.
3. In the following fields, specify:
   - Main class. Type `play.server.Server`
   - VM options. Type `-Dapplication.path="."`
   - Working directory. Specify your Play application root directory.
4. If necessary, change the run configuration name and click OK.

Now, to execute this run configuration, you can, for example, use Run | Run. (For more information, see Running Applications .)

Running Play application tests

To run tests for your Play application, you can use the `play test` command:

1. Start the play command-line utility (Tools | Play with Playframework).
2. In the Run tool window, after `play` , type `test` and press `Enter` .
3. Open a Web browser and go to `http://localhost:9000/@tests` to run the tests.

To be able to use an IntelliJ IDEA run/debug configuration for running your tests:

- Add `<play_dir>\modules\testrunner\lib\play-testrunner.jar` to the dependencies of your module. (See an example of the procedure to be used.)
- Modify the run configuration for running the application : in the VM options field, after `-Dapplication.path="."` , type space and then type `-Dplay.id=test`
  So, finally, the VM options field will contain:

```
-Dapplication.path="." -Dplay.id=test
```

(If you want a separate run configuration for the tests, you can create a copy of the existing run configuration ( ) and then modify that copy accordingly.)

Debugging a Play application: process overview

1. Create a run/debug configuration for debugging . This should be a Remote type of configuration; the port specified in this configuration should correspond to a Java debugger port (`8000` by default).
2. Set one or more breakpoints in your code. See Using Breakpoints .
3. If you are going to use an IntelliJ IDEA run configuration for starting the application, modify the corresponding run configuration , or create a new one.
4. Run the application by using the `play run` command (without any options and arguments) or by executing the corresponding run configuration.
5. Start the run/debug configuration intended for debugging.
6. In a Web browser, go to `http://localhost:9000` and, by using the application, try to reach a breakpoint.
7. When a breakpoint is reached, switch to IntelliJ IDEA and scrutinize the suspicious code fragments.
8. Continue debugging unless you localize the problem in your code.

See also, Debugging .

Creating a run/debug configuration for debugging

1. Open the Run/Debug Configurations dialog (e.g. Run | Edit Configurations).
2. Click + (`Alt+Insert`) and select Remote.
3. In the Port field, specify `8000` . (This is the port to which a Java debugger will connect. By default, the Play framework uses the port `8000` .)
4. If necessary, change the run/debug configuration name and click OK.

Modifying the run configuration for running the application

If you are using an IntelliJ IDEA run configuration for running your Play application , this configuration has to be modified so that a Java debugger could connect to the running application. This is done by adding the corresponding JVM options. The necessary options can be copied from the run/debug configuration intended for debugging.

1. Open the Run/Debug Configurations dialog (e.g. Run | Edit Configurations).
2. In the left-hand pane of the dialog, select the run/debug configuration intended for debugging.
3. Click to the right of the upper field containing JVM command-line arguments. As a result, the field contents are copied to the clipboard.
4. Select the run configuration for running the application, and paste the options into the VM options field. This field should finally contain `-Dapplication.path="."` , then a space and then the options you've just pasted. Here is an example:

```
-Dapplication.path="." -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000
```

(If you are using the same run configuration for running the application and the tests , you may need to delete `-Dplay.id=test` .)

5. Click OK.

Plugin Development Guidelines

In this section:

- Plugin Development Guidelines
  - Before you start
  - Developing plugins
- Configuring IntelliJ Platform Plugin SDK
- Creating a Project for Plugin Development
- Running and Debugging Plugins
- Preparing Plugins for Publishing
- Viewing PSI Structure

Before you start

1. Download and install a JDK.
2. Make sure that the Plugin DevKit plugin is enabled, see Enabling and Disabling Plugins .
3. Optionally, download the IntelliJ IDEA Community Edition sources. This will make debugging your plugins much easier.

Developing plugins

To develop an IntelliJ IDEA plugin

Follow these general steps:

1. Start IntelliJ IDEA.
2. Configure the IntelliJ Platform Plugin SDK .
3. Create a project for plugin development .
4. Create the necessary source elements and write the source code.
5. Run and debug your plugin.
6. Prepare the plugin for publishing .

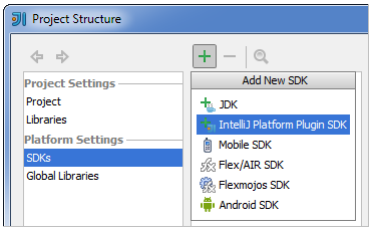
For more information on IntelliJ IDEA plugin development, refer to the Writing Plug-ins page.

Configuring IntelliJ Platform Plugin SDK

To be able to develop plugins for IntelliJ IDEA, you have to configure an IntelliJ Platform Plugin SDK . You can do that separately - as described on this page, or when creating a project .

To configure the IntelliJ Platform Plugin SDK

1. Open the Project Structure dialog (e.g. `Ctrl+Shift+Alt+S`).
2. Under Platform Settings, select SDKs and then click + to define a new SDK.
3. Select IntelliJ Platform Plugin SDK from the Add New SDK list.



4. If you haven't defined a JDK in IntelliJ IDEA yet, a dialog is shown suggesting that you do that now. Click OK in that dialog, and then select the installation folder of the desired JDK in the dialog that opens .
5. In the dialog that opens , select the installation folder of the necessary IntelliJ IDEA version. (An IntelliJ IDEA installation acts as an IntelliJ Platform Plugin SDK).
6. In the Select internal Java platform dialog, select the desired JDK from the list and click OK. (Listed in this dialog are the JDKs defined in IntelliJ IDEA.)
7. If necessary, change the path to the sandbox in the Sandbox Home field. (The sandbox is a folder where IntelliJ IDEA will copy your plugins for debugging.)
8. Optionally, configure the source path for the IntelliJ IDEA Community Edition sources (if you are going to use them and have them available on your computer). To do that, select the Sourcepath tab and click +. In the dialog that opens , select the folder containing the sources and click OK. Click OK in the Detected Source Roots dialog.
9. Click OK to save the changes and close the Project Structure dialog.

Now, the IntelliJ Platform Plugin SDK is in the list of available SDKs, so in the future, when necessary, you will be able to simply select it from the corresponding lists.

Creating a Project for Plugin Development

To develop a plugin for IntelliJ IDEA, you need a project with a plugin module .

To create a new project with a plugin module

1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen . Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.

2. On the first page of the wizard , in the left-hand pane, select IntelliJ Platform Plugin.

3. In the right-hand part of the page:
   - Specify the SDK to be used.
     If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select the installation folder of the desired IntelliJ IDEA version. (An IntelliJ IDEA installation acts as an IntelliJ Platform Plugin SDK.) If asked to specify a JDK first, do so.
   - If necessary, enable Groovy support by selecting the corresponding check box. If the necessary Groovy version is already defined in IntelliJ IDEA as a library , select that library from the list. Otherwise, click Create and select the installation folder of the desired Groovy version in the dialog that opens.
   - If necessary, enable SQL support by selecting the corresponding check box. Select the SQL dialect to be used by default from the list.

   Click Next.

4. Specify the name and location settings for your project and module. For more information, see Project Name and Location .
   Click Finish.

When a project is created, populate it with new actions , components, etc. To create the necessary items, use the New menu. To access this menu, select the required location, package, etc. in the Project tool window and then do one of the following:

- Select File | New in the main menu.
- Select New from the context menu.
- Press `Alt+Insert`.

Running and Debugging Plugins

To run or debug your plugin, you should create the corresponding Run/Debug configuration first.

To create a Run/Debug configuration for a plugin

1. On the main menu, choose Run | Edit Configurations.
2. In the Run/Debug Configuration dialog , click +, or press `Insert`, and select Plugin.
3. Specify the settings as necessary and click OK.

To run or debug a plugin

Depending on whether you are going to run or debug your plugin:

- To run the plugin, select Run | Run from the main menu, or press `Shift+F10`.
- To debug the plugin, select Run | Debug from the main menu, or press `Shift+F9`.

As a result, IntelliJ IDEA will start an instance of itself. This will be the instance in which your plugin will be available.

Preparing Plugins for Publishing

When your plugin is ready, you might want to upload it to the JetBrains Plugin Repository (http://plugins.intellij.net) or to the repository of your own . Yet, before uploading a plugin, you need to create a ZIP or JAR plugin archive. This section provides instructions for creating such an archive.

To create a plugin archive

- Right-click the plugin module in the Project view and select Prepare Plugin Module <module-name> For Deployment in the context menu.
  As a result, IntelliJ IDEA will create the necessary archive.

Note If the plugin module does not depend on libraries, a JAR archive will be created. Otherwise, a ZIP archive will be created that will include all the necessary libraries.

Viewing PSI Structure

If you want to explore the internal structure of source code as it is interpreted by IntelliJ IDEA, use the PSI viewer.

Important note

PSI viewer command is only available when there is at least one plugin module in project.

If you want this information to be available for any project, add the following line to the file `bin/idea.properties` under the product installation:

```
idea.is.internal=true
```

To view PSI structure of the source code

1. On the main menu, choose Tools | View PSI Structure.
2. In the PSI Viewer dialog box, type or paste the fragment of source code to be analyzed in the Text area, select file type, and specify the other options.
3. Click Build PSI Tree, and preview the generated PSI tree in the PSI Structure pane.

   If the source code in the Text area has been changed, click the same button to refresh preview.

RESTful WebServices

In this section:

- RESTful WebServices
  - Introduction
  - Developing RESTful Web services
- Preparing for REST Development
- Coding Assistance for REST Development
- Testing RESTful Web Services

Introduction

IntelliJ IDEA lets you develop, debug and test RESTful Web Services . The JSR-311 specification version 1.0 and the Jersey reference implementation are supported.

Because RESTful Web services can be of various types, there is no definite workflow to develop them. In this section we will outline some specific tasks that can be performed when developing a REST application in IntelliJ IDEA.

Developing RESTful Web services

To develop a RESTful Web service, follow these general steps

1. Make sure that the RESTful Web Services plugin is enabled. Create a new project or module for the service development, or enable the RESTful Web Services development support for an existing module. For more information, see Preparing for REST Development .
2. Populate the RESTful Web service module with the necessary classes and methods, possibly using the generated sample class .
3. Configure the artifacts to deploy.
4. Create a run configuration . On the Deployment tab, specify the artifacts to deploy.
5. Deploy the Web service on an HTTP server and start the server.

   Tip If you are using the sample service created with the module, choose Run | Run or press `Shift+F10` . IntelliJ IDEA creates a local HTTP server, starts it, whereupon deploys and runs the Web service.
6. Run the application locally or on a remote host.
7. Test the RESTful Web service .

Preparing for REST Development

At the IntelliJ IDEA level, the RESTful Web Services development support is based on the Java EE: RESTful Web Services (JAX-RS) plugin . This plugin is bundled with IntelliJ IDEA and enabled by default.

At a module level, the service development support can be enabled when creating a project, adding a new module to an existing project, and also for an existing module.

- Making sure that the RESTful Web Services plugin is enabled
- Enabling REST support when creating a project
- Enabling REST support when adding a module to a project
- Enabling REST support for an existing module

Making sure that the RESTful Web Services plugin is enabled

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand part of the dialog, select Plugins.
3. In the right-hand part of the dialog, on the Plugins page , type `restful` in the search box. As a result, only the plugins whose names and descriptions contain `restful` are shown in the list of plugins.
4. If the check box to the right of Java EE: RESTful Web Services (JAX-RS) is not selected, select it.
5. Click OK in the Settings dialog.
6. If suggested, restart IntelliJ IDEA.

Enabling REST support when creating a project

1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen . Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.
2. On the first page of the wizard , in the left-hand pane, select Java Enterprise . In the right-hand part of the page, specify the JDK to be used and select the Java EE version to be supported.
3. Under Additional Libraries and Frameworks, select the Restful Web Service check box.
4. In the lower part of the page, if necessary, select the Generate server code sample and the Generate client code sample check boxes. If you do so, Java classes with a `main()` method will be created for the server and client parts of your application.
5. You'll need a library that implements the JAX-RS API. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the files that implement the JAX-RS API. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

   Click Next.
6. Specify the name and location settings for your project and module. For more information, see Project Name and Location .
   Click Finish.

Enabling REST support when adding a module to a project

1. Open the project you want to add a module to, and select File | New | Module.
   As a result, the New Module wizard opens. (There are also other ways of accessing this wizard, see Opening the New Module wizard.)
2. On the first page of the wizard , in the left-hand pane, select Java Enterprise . In the right-hand part of the page, select the Java EE version to be supported.
3. Under Additional Libraries and Frameworks, select the Restful Web Service check box.
4. In the lower part of the page, if necessary, select the Generate server code sample and the Generate client code sample check boxes. If you do so, Java classes with a `main()` method will be created for the server and client parts of your application.
5. You'll need a library that implements the JAX-RS API. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the files that implement the JAX-RS API. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Click Next.

6. Specify the name and location settings for your module. For more information, see Module Name and Location .
   Click Finish.

Enabling REST support for an existing module

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the module of interest and select Add Framework Support.
3. In the left-hand pane of the Add Frameworks Support dialog that opens, select the RESTful Web Service check box.
4. In the right-hang part of the dialog, if necessary, select the Generate server code sample and the Generate client code sample check boxes. If you do so, Java classes with a `main()` method will be created for the server and client parts of your application.
5. You'll need a library that implements the JAX-RS API. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the files that implement the JAX-RS API. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.
6. Click OK in the Add Frameworks Support dialog.

Coding Assistance for REST Development

Besides the common Web services support , IntelliJ IDEA provides the following facilities for developing RESTful Web services:

- Generating a sample Web service and deploying it locally
- Code completion for MIME type.
- Code inspections and quick fixes.
- REST Client tool window for testing RESTful Web services.

Generating a Sample Web Service

When you create a module for developing a RESTful WebService, you can have sample code of the server side generated. IntelliJ IDEA creates a package with a `HelloWorld` class that has the following methods:

- The `main()` method that creates an instance of a local HTTP server and starts it.
- The `getClichedMessage()` method that returns a cliched message. The method is supplied with the `@Path` , `@GET` , and `@Produces` annotations.

Use this class as an example or populate it as necessary to develop and test your RESTful Web service. To start the sample Web service, choose Run | Run or press `Shift+F10`. IntelliJ IDEA creates a local HTTP server, starts it, whereupon deploys and runs the Web service.

Code Inspection and Quick Fixes

IntelliJ IDEA supports code inspections and suggests quick fixes in the following cases:

Issue	Default Quick Fix
Inconsistency between a method annotation and the method return type: a `@GET` annotated method of the void return type.	Change to String.
Inconsistency between the class type and the type of a method within the class: a non-resource class contains resource methods.	Remove `@GET` , `@POST` , `@PUT` , or `@DELETE` annotation from the method declaration.
A Root Resource Class does not contain resource methods.	Remove `@Path` annotation.
Different Root Resource Classes have the same `@Path` annotation.	Remove `@Path` annotation.
A `@Consumes` or a `@Produces` annotation uses a wrong MIME type.	Change to */*.

REST Client Tool Window

Dedicated REST Client tool window supports:

- Constituting URL addresses semiautomatically from the specified Deployment point and the `@Path` annotation.
- Submitting requests to the server.
- Displaying server responses in the Response tab.
- Code integration between the Java code and the contents of the REST Client window controls.

Change in the Java code	Change in the REST Client tool window
`@Path` annotation is updated.	The contents of the Path to resource drop-down list change.
`@Produces` annotation is updated.	The contents of the Accept drop-down list change.

Testing RESTful Web Services

Testing RESTful Web Services is supported via the REST Client bundled plugin, which is by default enabled. If not, activate it in the Plugins page of the Settings dialog box.

There are two main use cases when you need to compose and run requests to a RESTful Web service:

- When you have developed and deployed a RESTful Web service and want to make sure it works as expected: that it is accessible in compliance with the specification and that it responds correctly.
- When you are developing an application that addresses a RESTful Web service. In this case it is helpful to investigate the access to the service and the required input data before you start the development. During the development, you may also call the Web service from outside your application. This may help locate errors when your application results in unexpected output while no logical errors are detected in your code and you suspect that the bottleneck is the interaction with the Web service.

Testing a RESTful Web service includes the following checks:

- That URL addresses are constituted correctly based on the service deployment end-point and the method annotations.
- That the generated server requests call the corresponding methods.
- That the methods return acceptable data.

IntelliJ IDEA enables you to run these checks from the REST Client tool window by composing and submitting requests to the local server, viewing and analyzing server responses .

If necessary, configure the Proxy settings on the HTTP Proxy page of the Settings dialog box.

On this page:

- Composing and submitting a test request to a Web service method
- Viewing and analyzing responses from a Web service
- Working with cookies
- Configuring Proxy settings
- Reusing requests

To compose and submit a test request to a Web service method

1. If you are going to test your own Web service, make sure it is deployed and running.

2. Choose Tools | Test RESTful Web Service. The REST Client dedicated tool window opens.

3. To have IntelliJ IDEA generate an authentication header which will be used in basic authentication , click the Generate Authorization Header button and in the dialog box that opens, specify your user name and password for accessing the target RESTful Web service through. Based on these credentials IntelliJ IDEA will generate an authentication header which will be used in basic authentication . Learn more at http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm .

4. Select the test request method from the HTTP method drop-down list. The available options are:
   - GET
   - POST
   - PUT
   - PATCH
   - DELETE
   - HEAD
   - OPTIONS

5. Provide the data to calculate the URL address of the target method:
   1. In the Host/port text box, type the URL address of the host where the Web service is deployed.
   2. From the Path drop-down list, choose the @path annotation that corresponds to the method the call to which you need to test.

      Tip IntelliJ IDEA supports integration between the contents of this list and Web service Java source code. Any changes to the `@Path` annotations are reflected in the contents of the Path to Resource list.

      To synchronize the contents of the Path drop-down list with the @Path annotations, click the Update resource paths from code button.

   You can enter the entire URL address of a method to test in the Host/port text box. Regardless of the chosen HTTP method, upon pressing IntelliJ IDEA will split the URL address into the host/port and the path to the method. The extracted relative path will be shown in the Path text box and the extracted parameters will be added to the list in the Request Parameters pane of the Request tab.

6. In the Header data pane, specify the technical data included in the request header . These data are passed through header fields and define the format of the input parameters (accept field), the response format (content-type field), the caching mechanism (cache-control field), etc.
   To add a field to the list, click Add, then specify the field name in the Name text box and the field value in the Value drop-down list.

   The set of fields and their values should comply with the Web service API. In other words, the specified input format should be exactly the one expected by the Web service as well as the expected response format should be exactly the one that the service returns.

   For `accept` , `content-type` , and some other fields IntelliJ IDEA provides a list of suggested values. Choose the relevant format type from the Value drop-down list.

7. Create a set of parameters to be passed to the target method and specify their values. Depending on the chosen request method, you can create a list of parameters in two ways:
   - For `GET` requests, specify the parameters to be passed as a query string inside the URL. Use the Request Parameters pane. By default, the pane shows an empty list with one line.
     - To add a parameter, click Add, then specify the name of the parameter in the Name text box and the value of the parameter in the Value drop-down list.
     - To delete a parameter from the list, select it and click Remove.
     - To suppress sending the specified query string parameters and disable the controls in the Request Parameters pane, press the Don't send anything

toggle button .

- To have the parameters passed to the target method inside a request message body, use the Request Body pane or have them inserted in the request from a local file. The Request Body pane is disabled when the `GET`, `DELETE`, `HEAD`, or `OPTIONS` request method is selected.
  - To specify the parameters explicitly, choose the Text option and type the parameters and values in the text box.
  - To have the parameters inserted from a text file, choose the File contents option and specify the file location in the File to send field.
  - To have a binary file converted and sent in the request, choose the File upload(multipart/form-data) option and specify the file location in the File to send field.

  In either case, the set of parameters and their types should comply with the Web service API, in particular, they should be exactly the same as the input parameters of the target method.

8. To submit a request to the server, click the Submit request button.
   Note that the server may lack certificate, or be untrusted.

> **Note** If a server is not trusted, IntelliJ IDEA shows a dialog box suggesting you to accept the server, or reject it. If you accept the server as trusted, IntelliJ IDEA writes its certificate to the trust store. On the next connect to the server, this dialog box will not be shown.

Viewing and analyzing responses from a Web service

- To view the response to the server request, switch to the Response tab. The tab is opened automatically when a response is received. By default, the server response is shown in the format, specified in the request header through the content-type field.
- To have the response converted into another format and opened in a separate tab in the editor, use the View as HTML, View as XTML, or View as JSON buttons.
- To view the technical data provided in the header of a Web service response, switch to the Response Headers tab.

Working with cookies

Using the dedicated Cookies tab, you can create, save, edit, and remove cookies, both received through responses and created manually. The name and value of a cookie is automatically included in each request to the URL address that matches the domain and path specified for the cookie, provided that the expiry date has not been reached.

The tab shows a list of all currently available cookies that you received through responses or created manually. The cookies are shown in the order they were added to the list. When you click a cookie, its details become editable and are displayed in text boxes. See REST Client Tool Window for details.

- No specific steps are required to save a cookie received through a response, all the cookies received from servers are saved automatically. To edit a received cookie, click the row with the cookie in the list and update the details that are now shown in editable text boxes.
- To create a cookie manually, click and specify the following:
  - The name and value of the cookie to be included in requests.
  - The domain and path the requests to which must be supplied with the name, value, and expiry date of the cookie.
- To remove a cookie from the list, select the row with the cookie and click .

Configuring Proxy settings

1. Click the Configure HTTP Proxy icon.
2. In the Proxy dialog that opens, specify the following:
   - Enter the Proxy host name and Proxy port number in the Proxy host and Proxy port text boxes respectively.
   - To enable authorization, check the Use authorization check box and type the User name and password in the relevant fields.

Reusing requests

During a IntelliJ IDEA session, IntelliJ IDEA keeps track or requests and you can run any previously executed request. You can also save the settings of a request in an XML file so they are available in another IntelliJ IDEA session. When necessary, you can retrieve the saved settings and run the request again.

- To rerun a request within a IntelliJ IDEA session:
  1. Click the Replay Recent Requests button.
  2. From the Recent Requests pop-up list, select the relevant request. The fields are filled in with the settings of the selected request.
  3. Click the Submit Request button.
- To save the settings of a request so they can be retrieved in another IntelliJ IDEA session, click the Export Request button. In the dialog box that opens, specify the name of the file to save the settings in and its parent folder.
- To run a request saved during a previous IntelliJ IDEA session:
  1. Click the Import Request button.
  2. In the dialog box that opens, select the relevant XML file. The fields are filled in with the settings of the selected request.
  3. Click the Submit Request button.

Scala

The Scala Plugin extends IntelliJ IDEA with the full-scale functionality for Scala development.

The Scala plugin support include the following features:

- Full Scala language support
- Syntax and error highlighting
- Sclala run/debug configurations , inspections , intention actions , and refactorings .

The Scala plugin also supports the following tools and frameworks:

- SBT
- SSP
- HOCON
- Play 2

Before you start working with the plugin make sure that the following prerequisites are met:

- The Scala SDK is downloaded and installed on your machine. IntelliJ IDEA supports the Scala versions 2.8 and later.
- The Scala plugin is downloaded and enabled in IntelliJ IDEA.

Creating and Running Your Scala Application

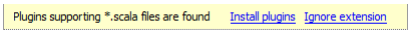
IntelliJ IDEA lets you create and run a Scala application.

– Before you start
– Creating a project
– Exploring the project structure
– Running your application
– Creating the object

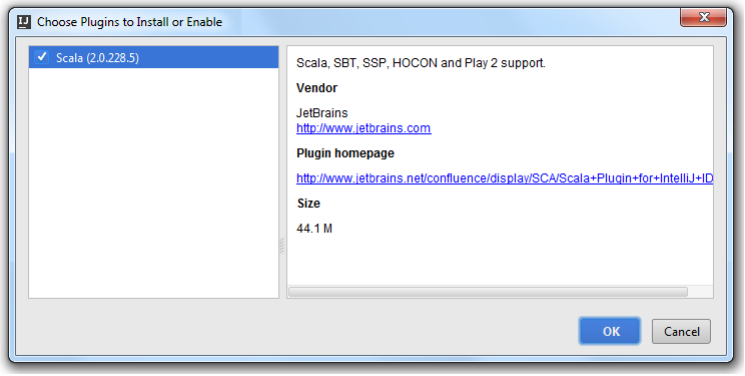
Before you start

By default, the Scala plugin is not bundled with IntelliJ IDEA.

If IntelliJ IDEA recognizes that you have `.scala` files in your project then it displays the appropriate pop-up suggesting to install the Scala plugin.



Click the link and in the dialog that opens, click OK.



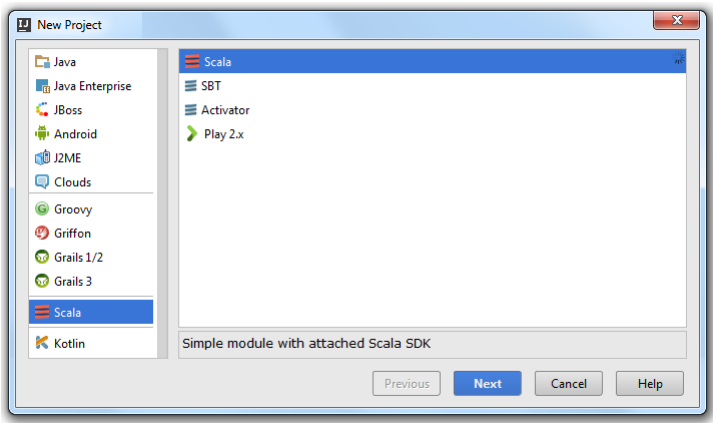
Alternatively, you can use Settings to download and enable the plugin in IntelliJ IDEA.

Creating a project with a Scala module

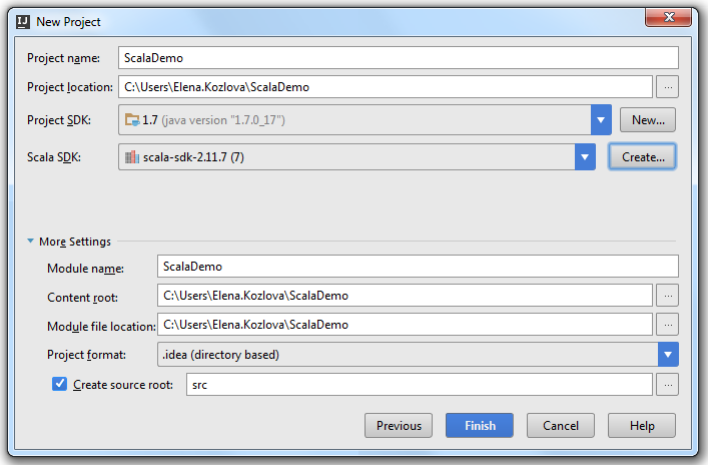
1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen. Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.
2. In the left-hand pane, select Scala.

   Note that if you select Java and in Additional Libraries and Frameworks , select Scala, you will create a Java project with a simple Scala module. However, the project will not include SBT, Play 2.x or Activator.
3. In the right-hand pane, select Scala.

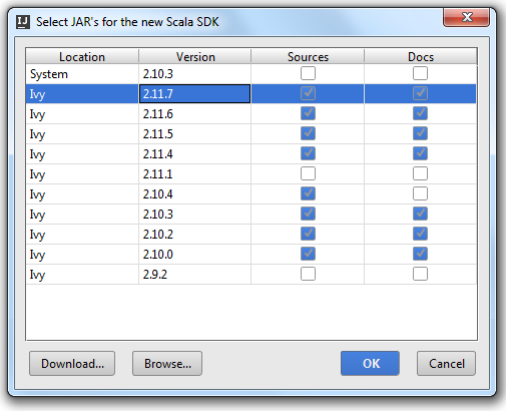
   You can also select SBT for creating an SBT project, Play 2.x for creating a project with the Play 2.x framework or Activator for creating a project with Typesafe Activator templates.



   Click Next.
4. On the next page of the wizard, specify project and module location settings.



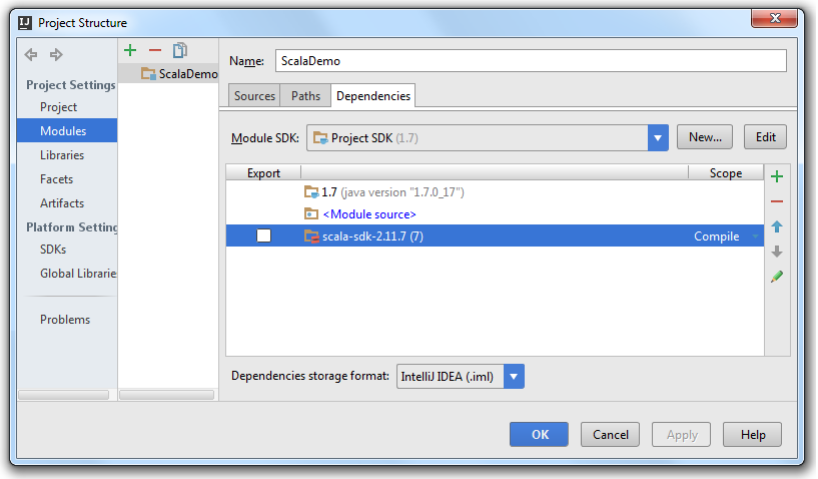
If the Scala SDK is not specified, click Create, select a necessary SDK version for downloading and click OK.



Click Finish.

Exploring your project structure

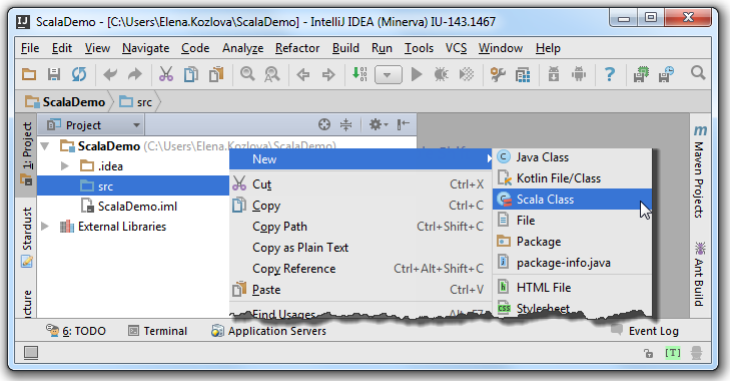
1. Open Project Structure .
2. On the left-hand pane, select Modules and on the right-hand side of the module page , you can check the project libraries, add the new ones, edit or delete the existing libraries.



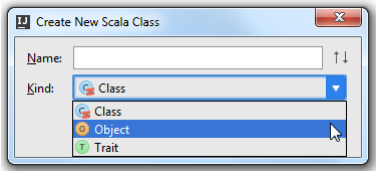
Click OK.

Creating an object

1. In the project tree, right-click a node in which you want to create your Scala object.
2. From the drop-down list, select Scala class.



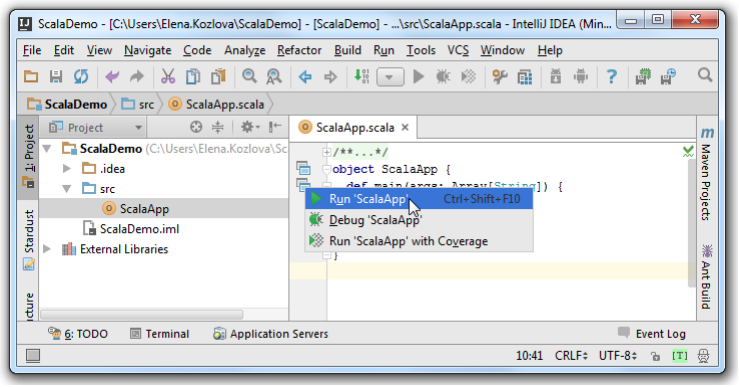
3. In the dialog that opens, in the field Name, enter the name of your Scala object and in the field Kind select either Object, Class or Trait.



Click OK.

Running your application

1. In the project tree, right-click the application and from the context menu select Run 'application name'. Alternatively, press `Shift+F10` to run the application. You can also right-click the icon in the left gutter and select Run 'application name' command.



Getting Started with Play 2.x

- Creating a project
- Importing a Play 2.x project
- Checking project settings
- Using code assistance
- Running a Play 2.x application
- Debugging a Play 2.x application

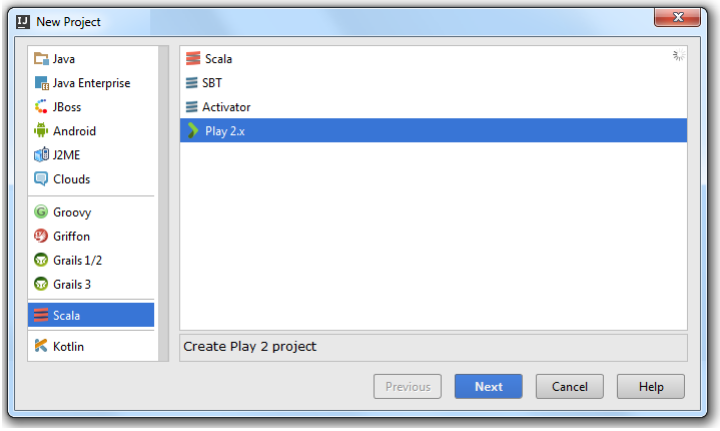
Creating a project

Before you create a project make sure that the Scala plugin is downloaded and enabled in IntelliJ IDEA.

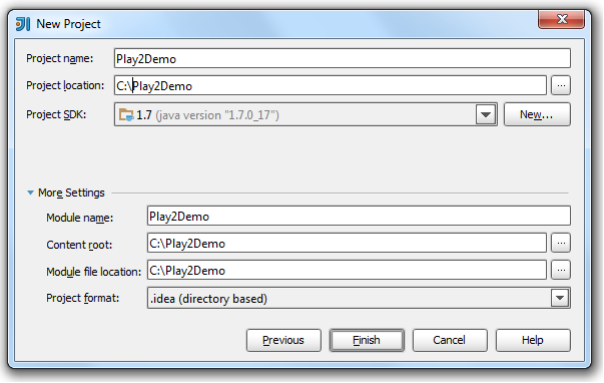
1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen. Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.
2. In the left-hand pane, select Scala.

   Note that for creating a Java project with the Play 2.x framework, you need to select Java.
3. In the right-hand pane, select Play 2.x.

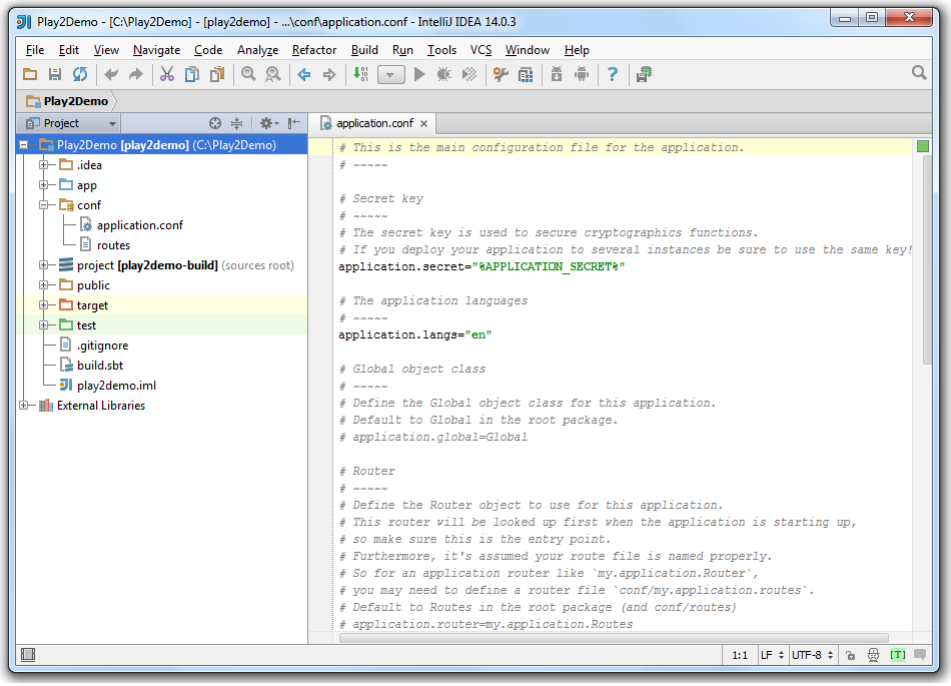
   You can also select Scala for creating a project with the Scala module, SBT for creating a project with the SBT module or Activator for creating a project with Typesafe Activator templates.



   Click Next.
4. On the next page of the wizard, specify project and module location settings.



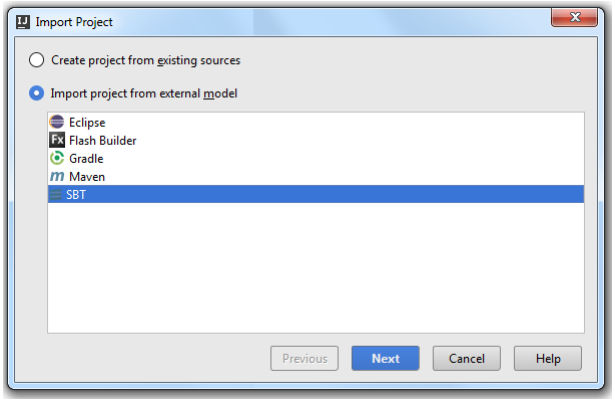
   Click Finish. The IDE will create an empty application.



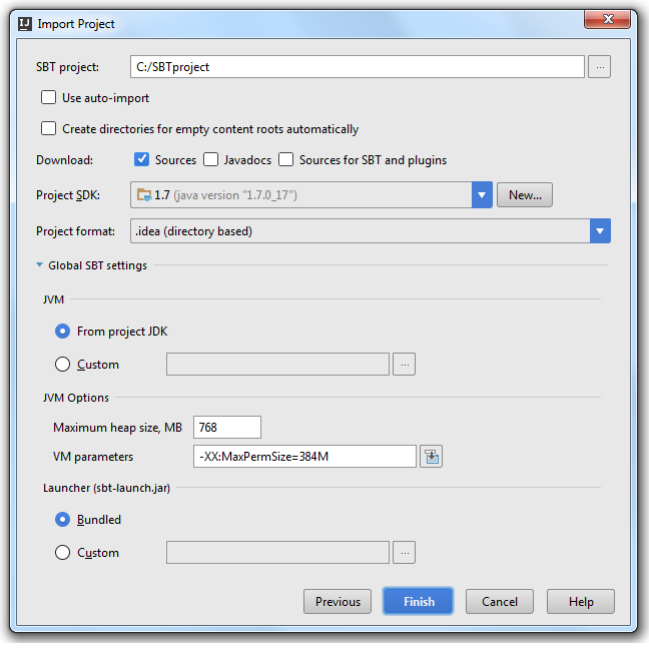
Importing a Play 2.x project

IntelliJ IDEA lets you import an existing Play 2.x project.

1. On the main menu, select File | New | Project from Existing Sources.
2. In the window that opens, select a file that you want to import and click OK. The Import project wizard opens.
3. On the first page of the wizard, select Import project from external model option and choose SBT project from the list. Click Next.

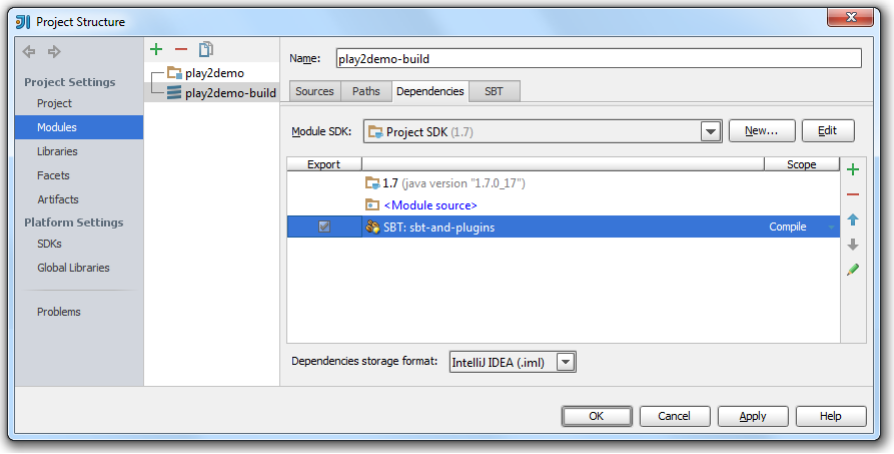


4. On the next page of the wizard, select SBT options and click Finish.

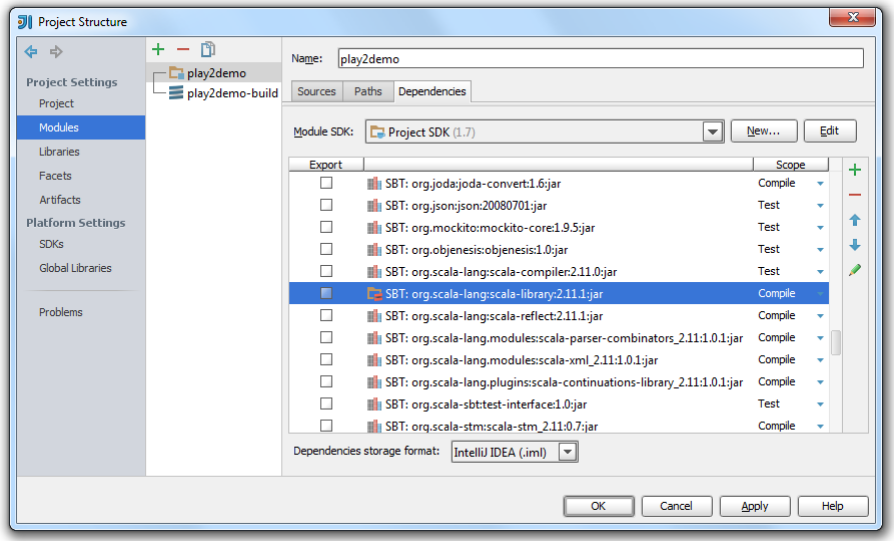


Checking project settings

1. On the main menu, select File | Project Structure. Alternatively, use icon on the Toolbar.
2. In the Project Structure dialog, check if module dependencies are resolved without warnings.

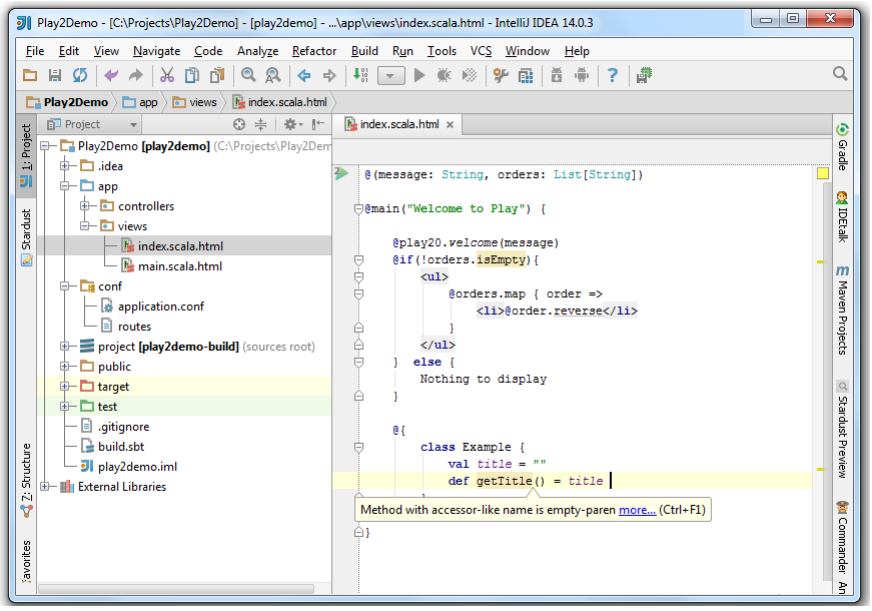


Also,ensure that the Scala compiler library is set.



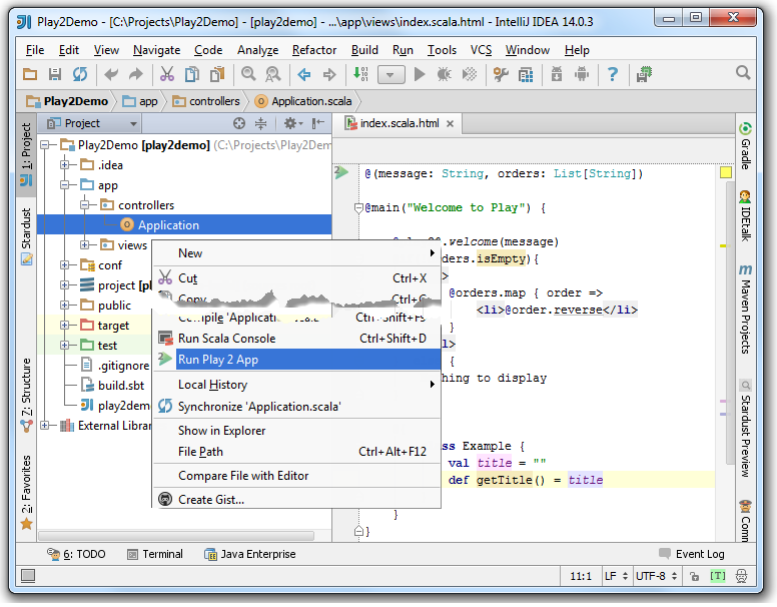
Using code assistance

When everything is set up, you can use code completion, navigation and on-the-fly code analysis features in your Play files. IntelliJ IDEA also supports code assistance for routes files and code inspections.



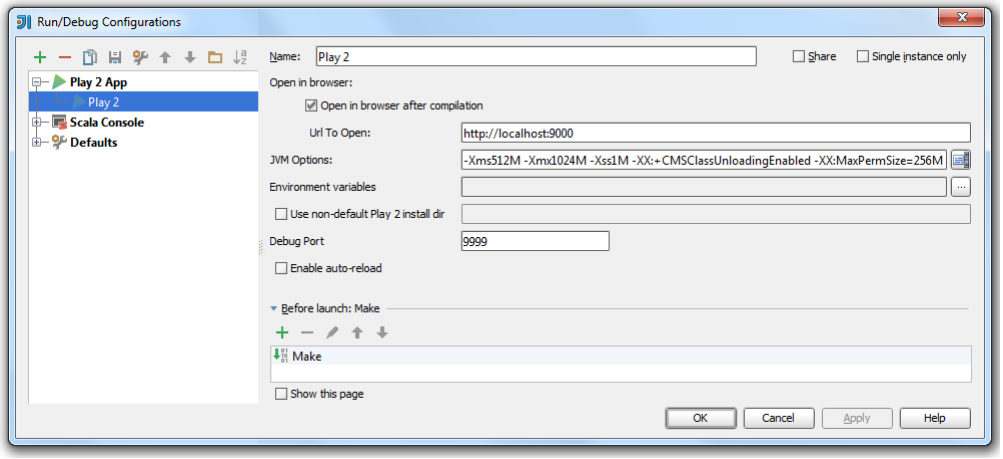
Running a Play 2.x application

1. In the project tool window, right-click the application.
2. On the context menu, select Run Play 2 App.



Debugging a Play 2.x application

1. On the main menu, select Run | Debug.
2. From the list that opens, select Edit Configurations.
3. In the dialog that opens, specify settings for debugging or use the default ones and press OK. IntelliJ IDEA will start a debugging session.



Getting Started with SBT

IntelliJ IDEA lets you create a new or import an existing project with a SBT module. SBT is a build tool for your Scala projects. You can also use SBT for Java projects.

– Before you start
– Creating a project
– Exploring the project structure
– Importing an SBT project
– Adding and refreshing a dependency in the SBT project
– Editing SBT project settings

Before you start

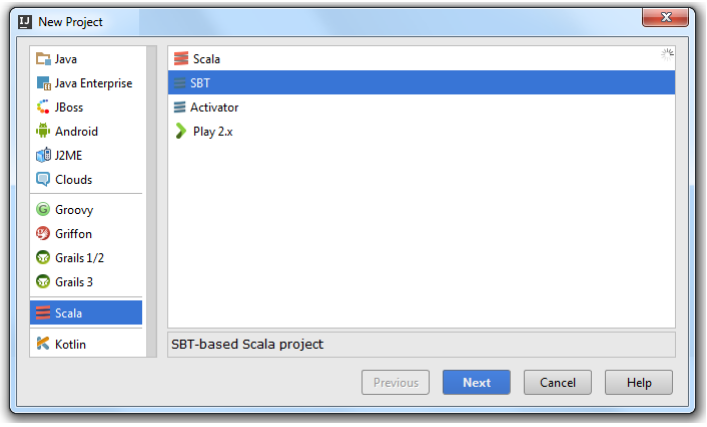
Before you start creating your SBT project, make sure that the Scala plugin is downloaded and enabled in IntelliJ IDEA.

Creating an SBT project

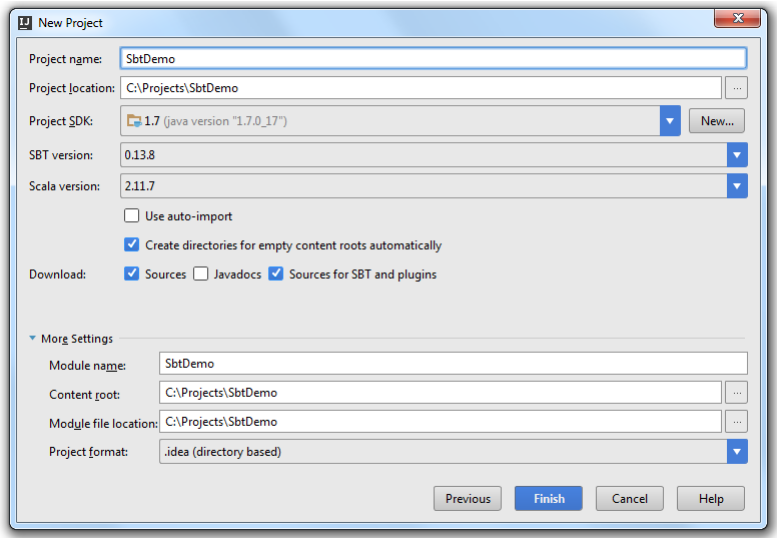
1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen. Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.
2. In the left-hand pane, select Scala.

   Note that for creating a Java project with an SBT module, you need to select Java.
3. In the right-hand pane, select SBT.

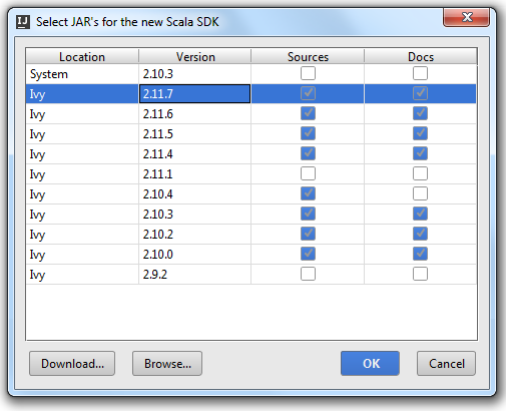
   You can also select Scala for creating a project with the Scala module, Play 2.x for creating a project with the Play 2.x framework or Activator for creating a project with Typesafe Activator templates.



   Click Next.
4. On the next page of the wizard, specify project and module location settings.



   If the Scala SDK is not specified, click Create, select a necessary SDK version for downloading and click OK.



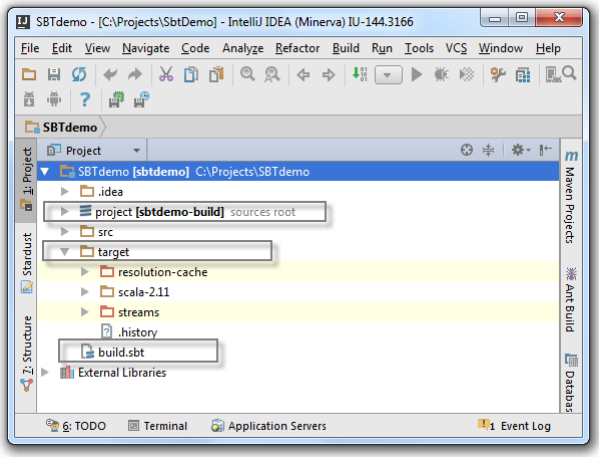
5. Specify the following additional options for your project:
   - Use auto-import - select this check box to resolve all the changes made to the SBT project automatically every time you refresh your project.
   - Create directories for empty content roots automatically - select this option to add the *src* directory to your project.
   - Download sources and docs - select this check box to download sources and docs for project dependencies.
   - Download SBT sources and docs - select this check box to download sources docs for SBT itself.

   Click Finish.

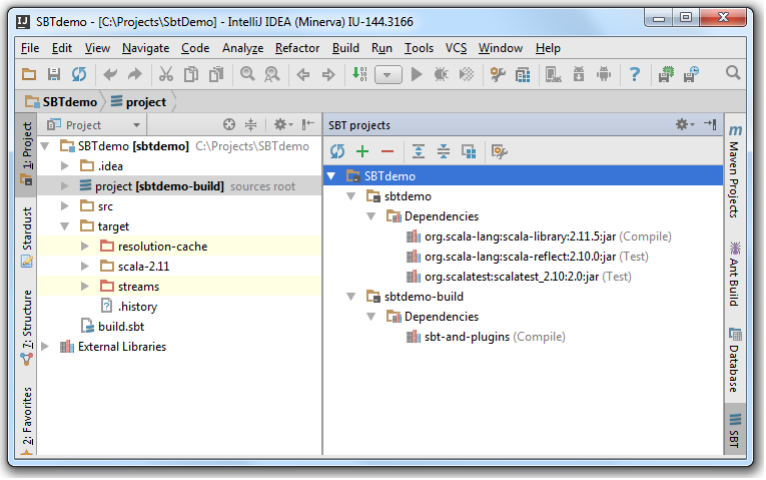
Exploring project structure

IntelliJ IDEA creates the following SBT-specific directories:

- project directory - this directory contains build-related information such as configuration classes.
- target directory - this directory contains compiler-related information such as generated objects and classes and is excluded by default.
- build.sbt file - in this file you can add dependencies, plugins, etc.



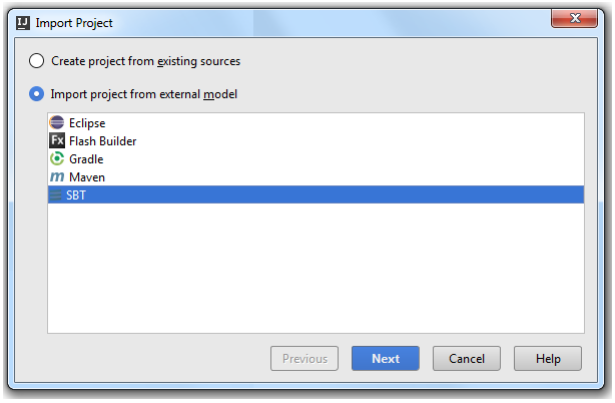
IntelliJ IDEA also creates an SBT Tool window. You can use this window for adding tasks, refreshing your SBT projects, attaching or detaching external projects.



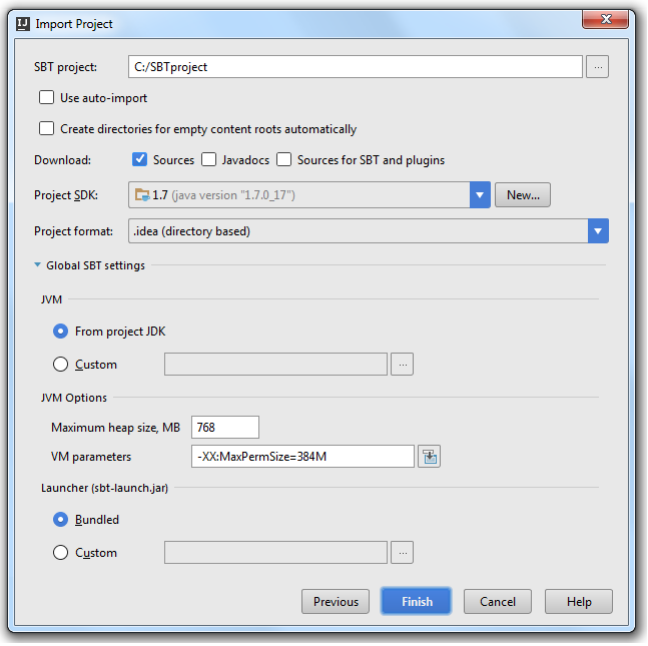
Importing an SBT project

IntelliJ IDEA lets you import an existing SBT project.

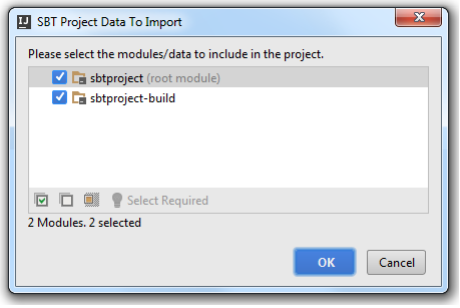
1. On the main menu select File | New | Project from Existing Sources.
2. In the window that opens, select a file that you want to import and click OK. The Import project wizard opens.
3. On the first page of the wizard, select Import project from external model option and choose SBT project from the list. Click Next.



4. On the next page of the wizard, select SBT options and click Finish.



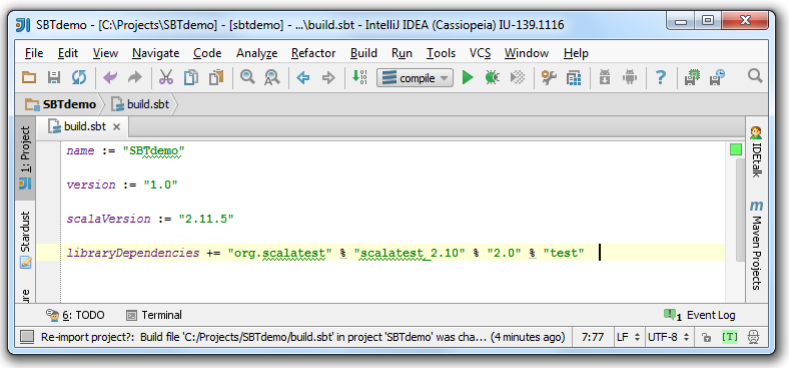
If you import a multi-module project, IntelliJ IDEA opens the SBT Project Data to Import dialog where you can select modules that you want to include in your project.



Adding and refreshing a dependency in SBT project

IntelliJ IDEA lets you add and refresh dependencies in your SBT project.

1. Open the `build.sbt` file and add a dependency.



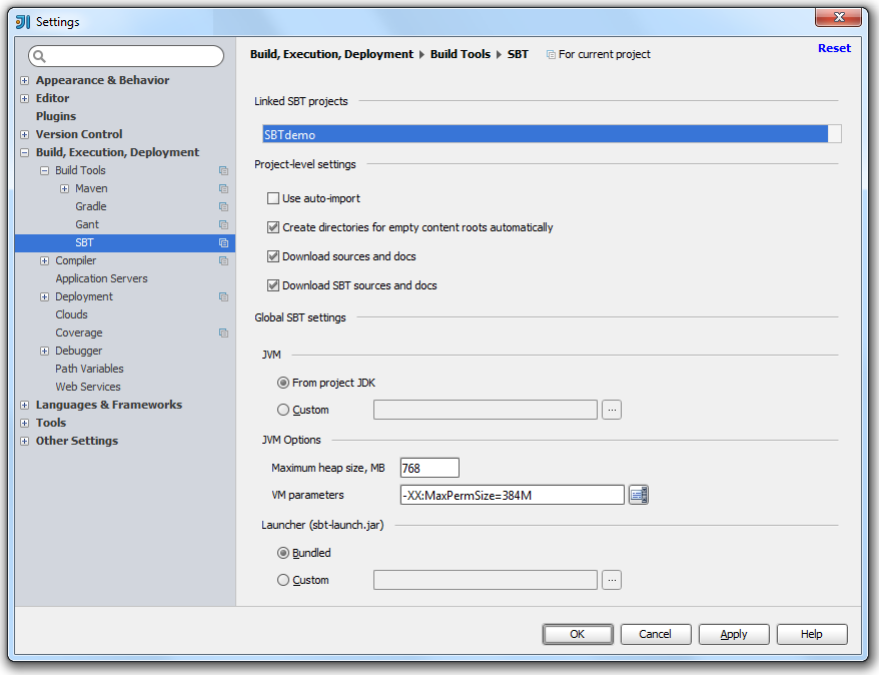
2. If you chose the Use auto-import option when you imported your project than your dependency is imported automatically.
3. If you didn't choose Use auto-import option then after you have added your dependency in the `build.sbt` file, press Refresh all SBT projects button in the SBT tool window to import the dependency. Note that you will also see the pop-up message suggesting to enable the auto-import option.

Editing SBT project settings

IntelliJ IDEA lets you edit its project's settings. This might be helpful if you, for example, didn't specify some options during the SBT project's import.

1. On the main menu, select File |Settings ('Preferences' for OSX) | Build, Execution, Deployment | Build Tools | SBT.

   The SBT page opens. Edit the settings and click OK.



Note The SBT settings that you specified through Project Structure dialog will be overwritten when you perform the next refresh of the project. We recommend you to use sbt-ide-settings plugin in order to preserve IDE-related settings. For further information, please see the plugin's README file.

Getting Started with Scala.js

IntelliJ IDEA lets you import or check out from VCS an existing Scala.js project. Scala.js compiles Scala code to JavaScript and lets you write your Web application entirely in Scala.

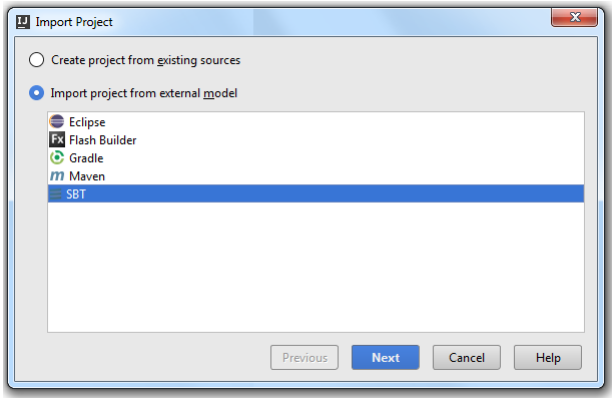
Before you start

Before you start importing your Scala.js project, make sure that the Scala plugin is downloaded and enabled in IntelliJ IDEA.

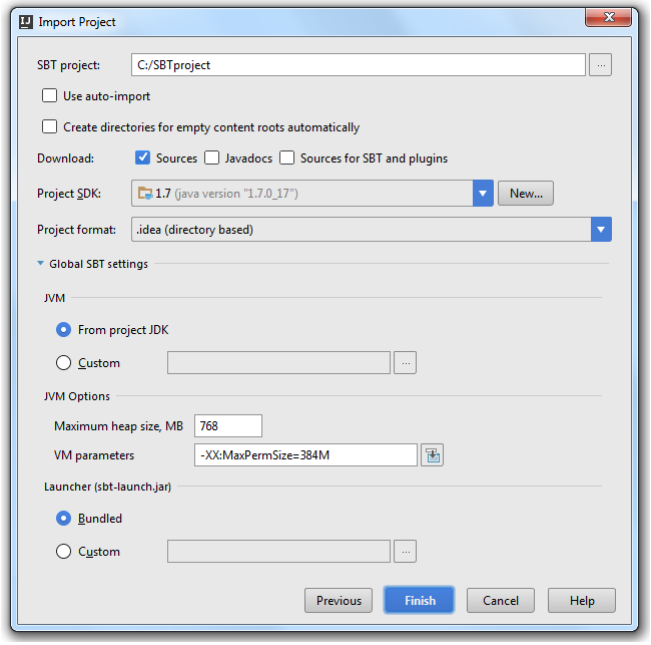
Importing a Scala.js project

IntelliJ IDEA lets you import an existing Scala.js project.

1. On the main menu select File | New | Project from Existing Sources.
2. In the window that opens, select a file that you want to import and click OK. The Import project wizard opens.
3. On the first page of the wizard, select Import project from external model option and choose SBT project from the list. Click Next.



4. On the next page of the wizard, select SBT options and click Finish.

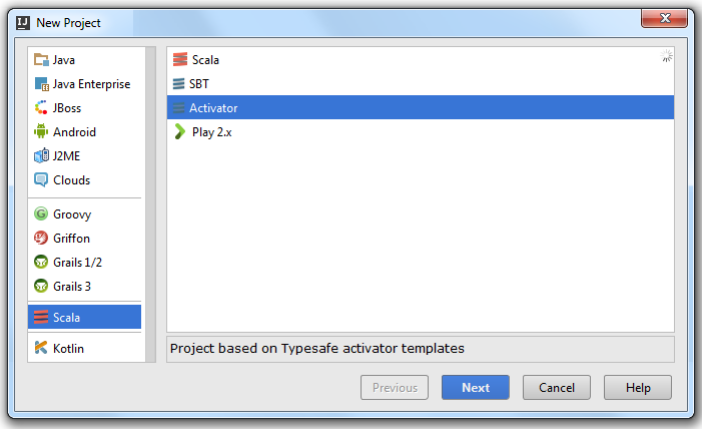


Creating a project

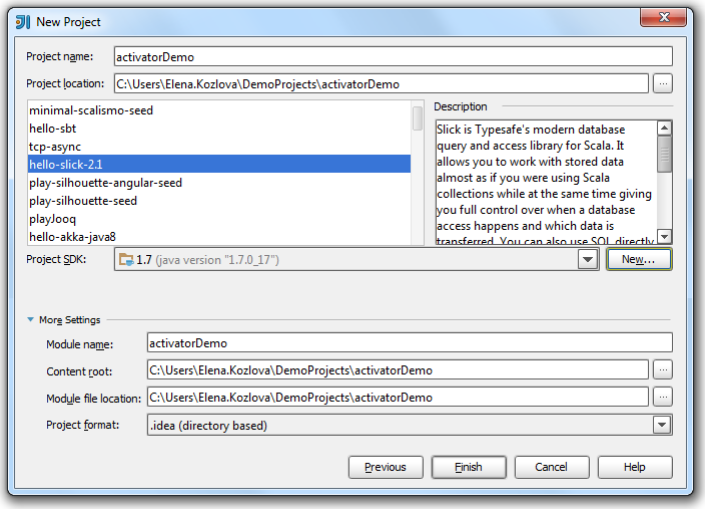
Before you create a project make sure that the Scala plugin is downloaded and enabled in IntelliJ IDEA.

1. If no project is currently open in IntelliJ IDEA, click Create New Project on the Welcome screen. Otherwise, select File | New | Project.
   As a result, the New Project wizard opens.
2. In the left-hand pane, select Scala.
3. In the right-hand pane, select Activator.

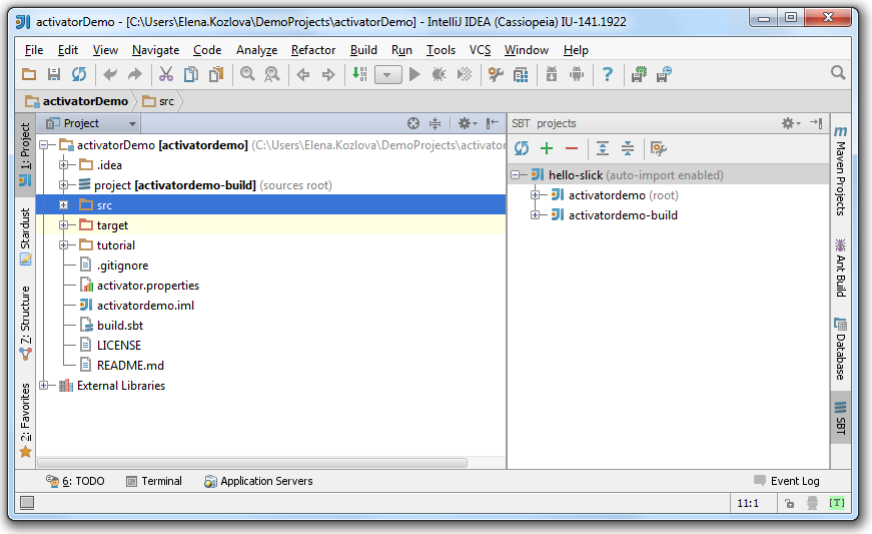
   You can also select Scala for creating a project with the Scala module, SBT for creating a project with the SBT module or Play 2.x for creating a project with the Play 2.x framework.



   Click Next.
4. On the next page of the wizard, specify your project's information, select a template application that you want to open and click Finish.



   The IntelliJ IDEA will create a project with the selected template application.



Working with Scala Worksheet

IntelliJ IDEA lets you create a Scala worksheet that enables you to evaluate your Scala code results in a special view of the Scala editor.

– Creating Scala Worksheet

– Running Scala Worksheet

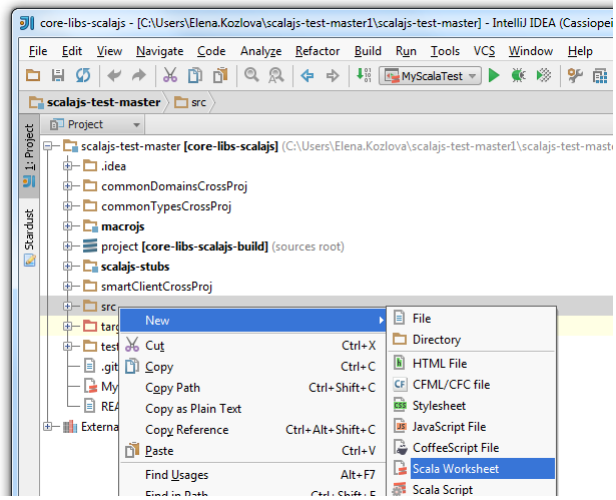
– Removing Worksheet's Results

– Copying Actions

– Configuring Worksheet Settings

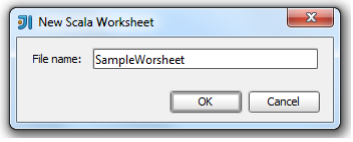
Creating Scala Worksheet

1. Right-click on your project and select New| Scala Worksheet.

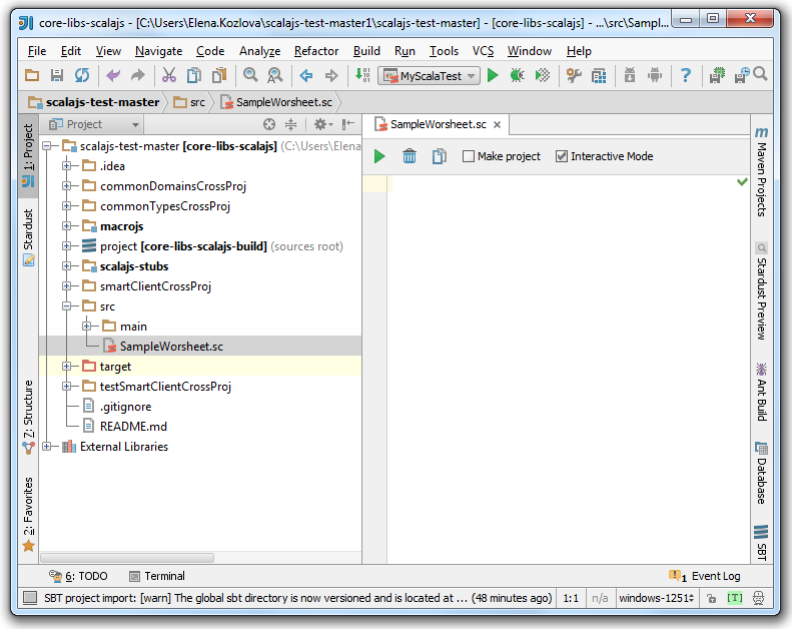


It is recommended to create your worksheet in src directory to simplify an inclusion of the project's classes in the classpath. That might be helpful for testing purposes.

2. In the New Scala Worksheet window type a name of your Scala worksheet and click OK.

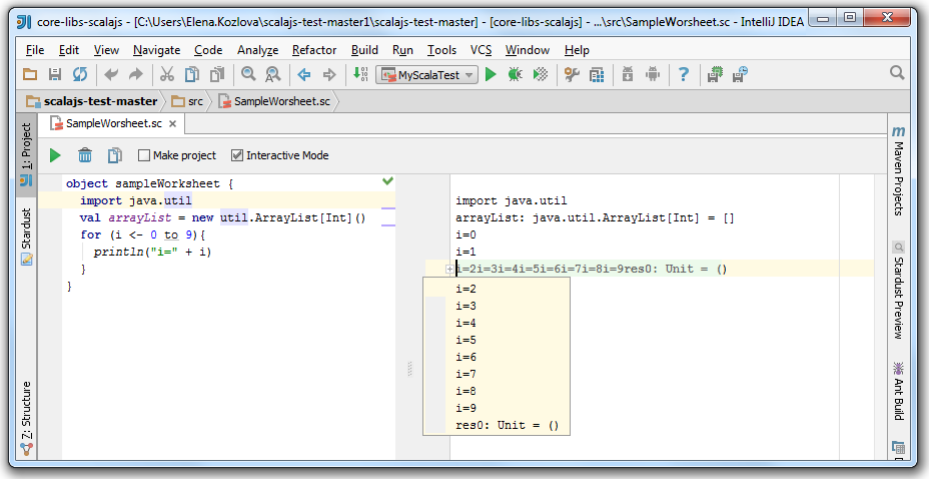


As a result, a file with .sc extension opens. Now you can enter your code.

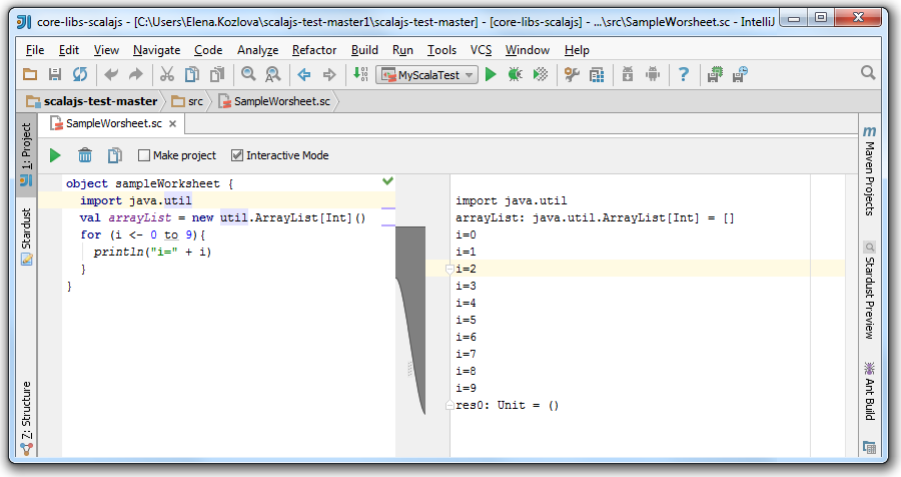


Running Scala Worksheet

1. Enter your code and press ▶ to see the results. If you select Interactive Mode check box then the code results will be displayed automatically.
   You can clear the Make project check box to improve evaluation performance. In this case, the automatic checking of project's changes is disabled. The results appear in the view on the right side of your code.
2. You can evaluate a Scala object.



You can fold the output without affecting your code on the left side and expand only that block output that matches a specific statement.

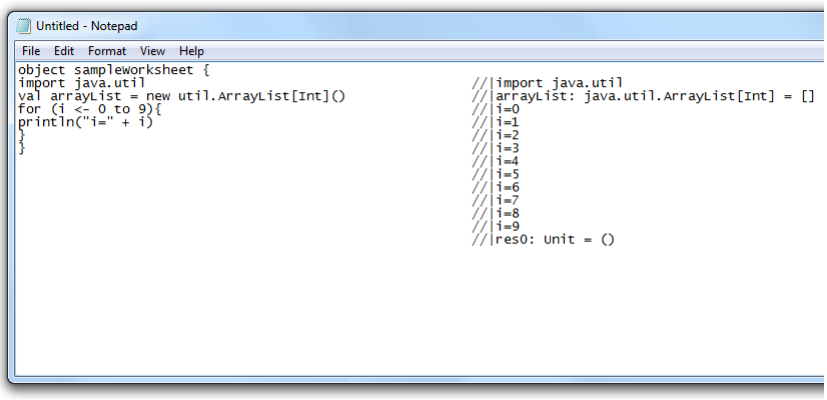


Removing Worksheet's Results

– Remove the worksheet's results easily by pressing icon.

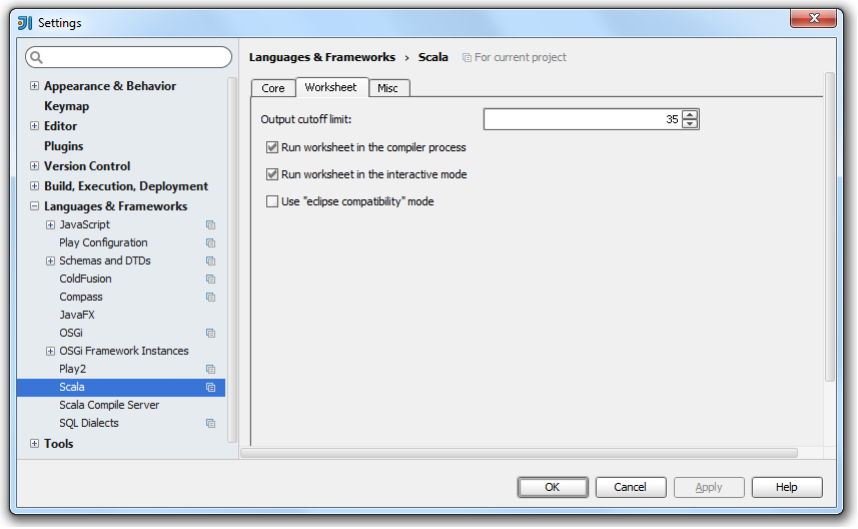
Copying Actions

– Copy your code and evaluation results into one file by clicking icon.



Configuring Worksheet Settings

1. Select File| Settings| Languages and Frameworks| Scala and click Worksheet tab.
2. On the Worksheet tab, you can set the output cutoff limit and configure in what mode you want to run the worksheet.


Settings
Languages & Frameworks › Scala
For current project
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
JavaScript
Play Configuration
Schemas and DTDs
ColdFusion
Compass
JavaFX
OSGi
OSGi Framework Instances
Play2
Scala
Scala Compile Server
SQL Dialects
Tools
Core
Worksheet
Misc
Output cutoff limit:
35
Run worksheet in the compiler process
Run worksheet in the interactive mode
Use "eclipse compatibility" mode
OK
Cancel
Apply
Help

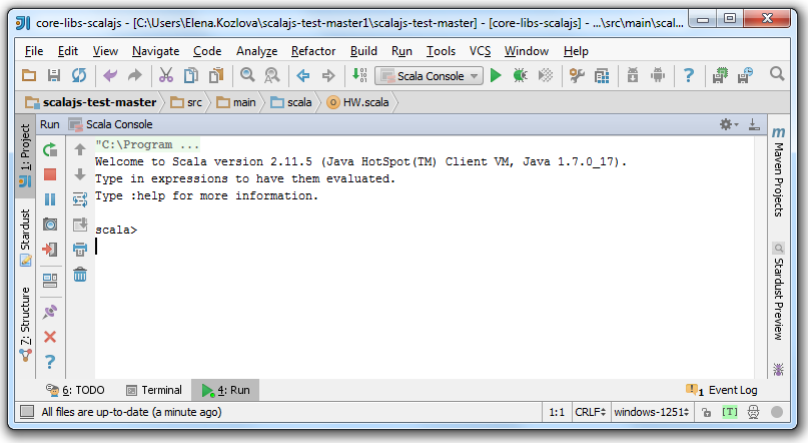
Working with Scala Console

IntelliJ IDEA lets you run Scala Console to quickly evaluate an expression or a code selection.

- Running Scala Console
- Evaluating Code in Scala Console
- Using Toolbars in Scala Console

Running Scala Console

- Right-click on your Scala object and select Run Scala Console from the drop-down list.
  The Scala Console window opens.

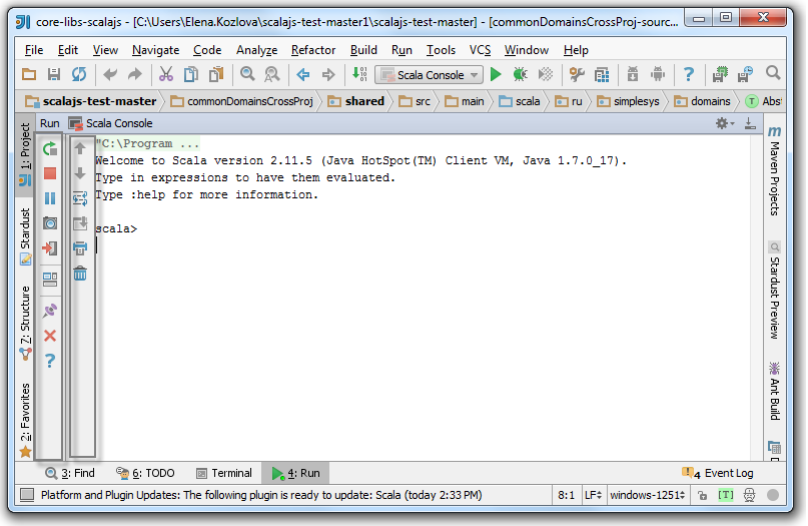


Evaluating Code in Scala Console

1. Type an expression in the console and evaluate it. You can also evaluate a code selection by sending it from the editor to the Scala console.
2. To see the result of the typed expression, press Ctrl+ Enter.
3. To send the code selection to Scala Console, select the code in the editor, press Ctrl+ Shift+ X (for Windows) or Meta+ Shift+ X (for Mac). Alternatively, right-click on the code selection and from the context menu select Send Selection To Scala Console to get the same result. Now, you can view the selected code and its evaluation in the console.

Using Toolbars in Scala Console

- You can use Run toolbar and Console toolbar options to rerun, stop, pause, or terminate the Scala console.



Working with Scala Tests

IntelliJ IDEA lets you create and run the following Scala tests:

– scalaTest

– spec2

– uTest2

IntelliJ IDEA supports the following methods in Scala tests:

– string constants

– constructions like immutable string variables

– concatenation

– substring

– stripSuffix

– stripPrefix

– toLowerCase

– replace

– trim

In this section:

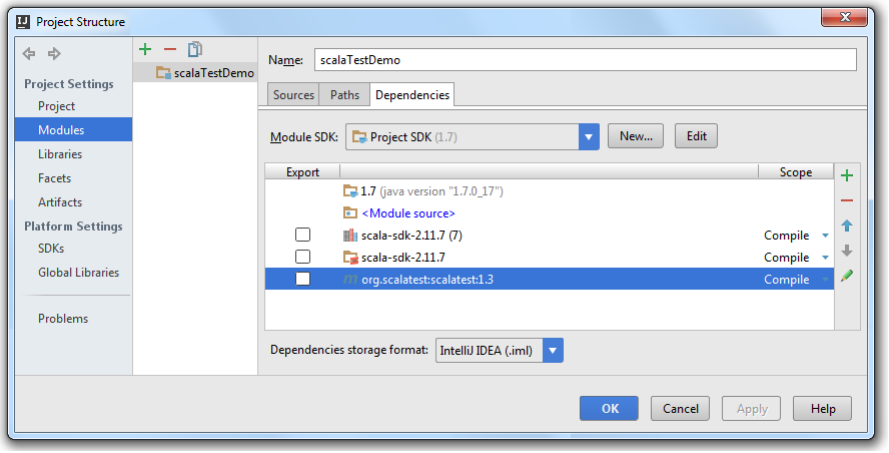
– Creating a Scala Test

– Running Scala Tests

– Viewing a Scala Test File Structure

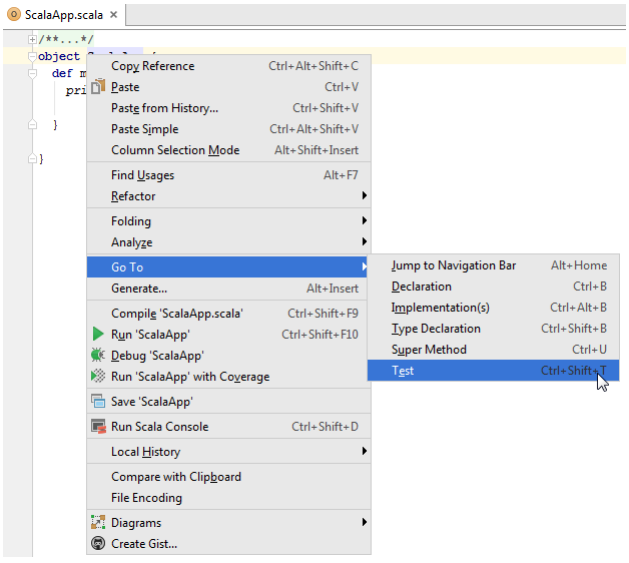
Creating a Scala Test

Before you start creating Scala tests, make sure you have the appropriate scala test dependency version added to your SBT project or you have the appropriate test library downloaded to your regular Scala project. You can use Project Structure dialog for these purposes.

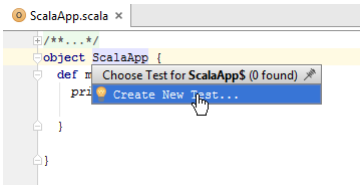


1. Open the class of interest in the editor.
2. Place the cursor within the line containing the class declaration.
3. On the context menu, choose Go to | Test.

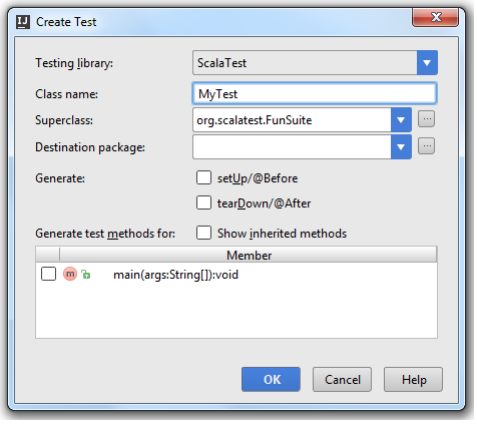
Note Alternatively, on the main menu, you can choose Navigate | Test.



4. If the target test doesn't exist, you are prompted to create it.

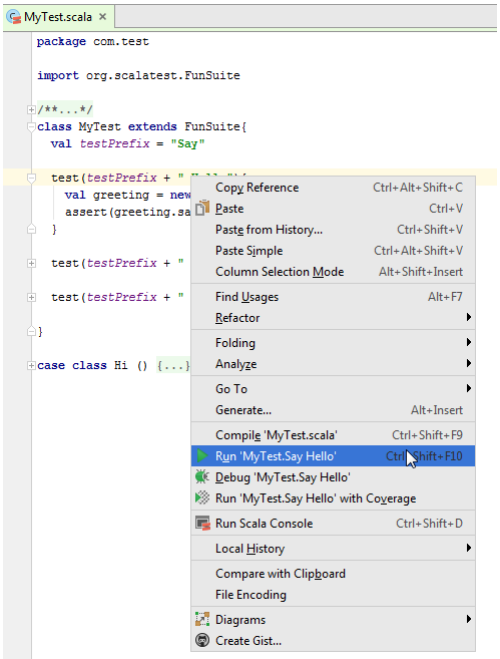


5. In the Create Test dialog specify the necessary information and click OK.

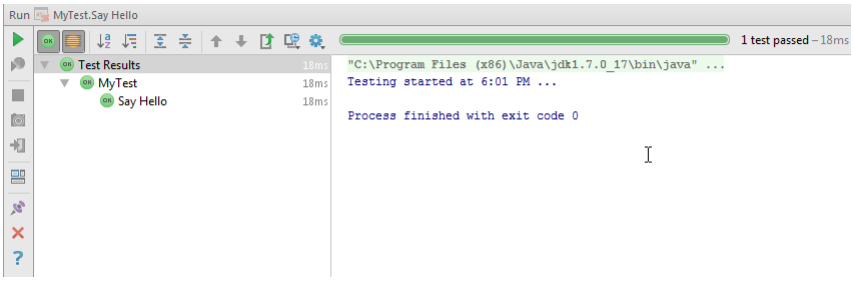


Running Scala Tests

1. Open your test file.
2. In the Editor enter your code, right-click on the class and from the context menu, select Run 'test name'.

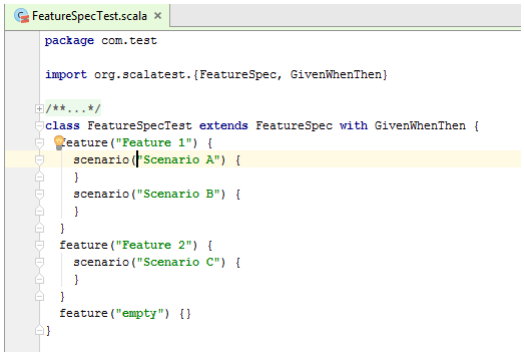


3. The result is displayed in the Test Runner tab of the Run tool window.

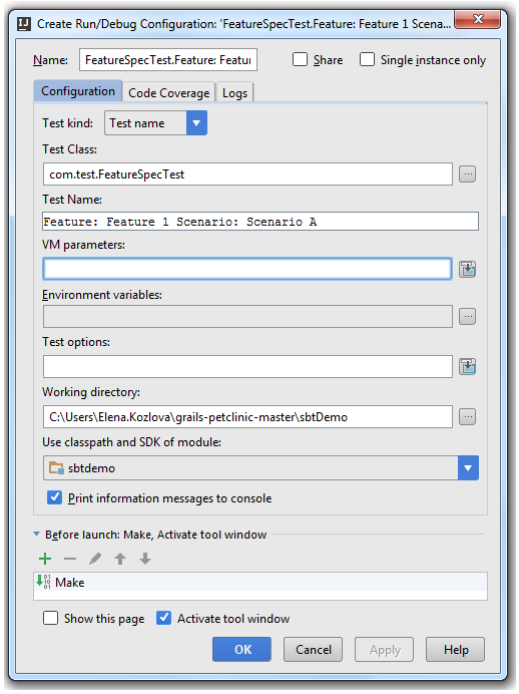


You can also run the tests inside the scope for scalaTest.

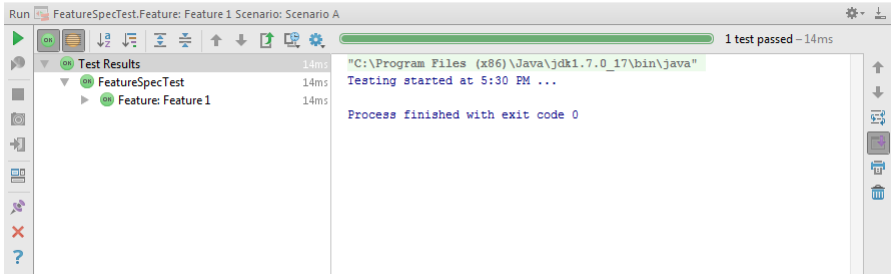
Check the following example:



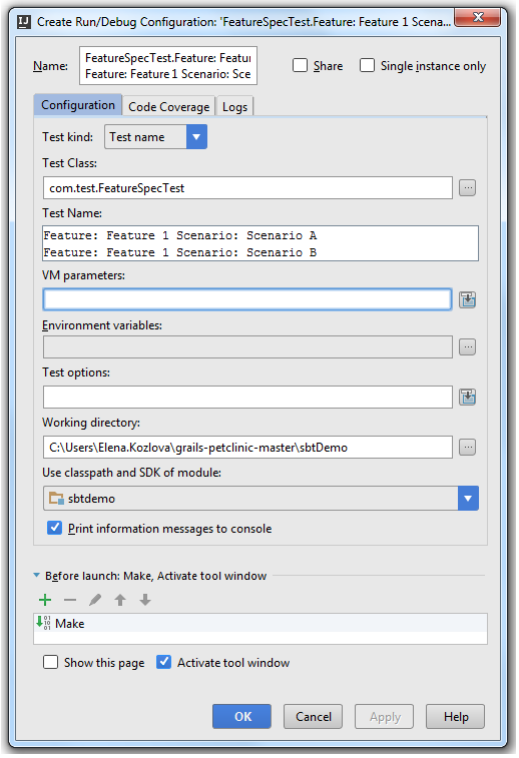
If you open context menu for one of the tests inside the scope, you can create a Run/Debug configuration just for the specified test inside that scope.



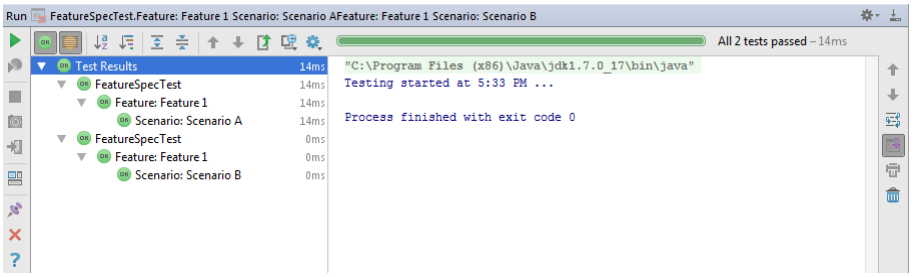
You can run the test and view the output inside the Run tool window.



If you open a context menu for the whole scope, you can create a Run/Debug configuration for all the tests inside the scope.

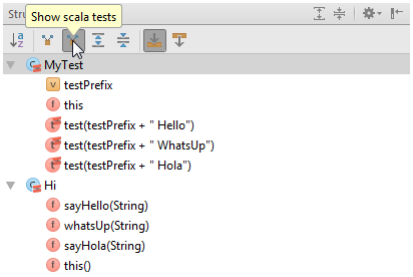


You can run the test for the whole scope and view the output inside the Run tool window.



Viewing a Scala Test File Structure

1. Click View | Tool Windows | Structure.
2. View the result in the Structure Tool Window.



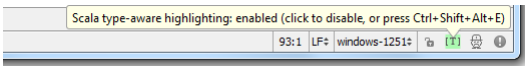
Working with Type-Aware Highlighting

Scala type-aware highlighting lets you get an error analysis based on the Scala type system before you start the compilation process.

- Enabling or Disabling Type-Aware Highlighting on the Project Level
- Enabling or Disabling Type-Aware Highlighting on the File Level
- Working with Type-Aware Highlighting in Editor
- Disabling Type-Aware Highlighting Locally

Enabling or Disabling Type-Aware Highlighting on the Project Level

- Enable or disable the type-aware highlighting for your project by clicking [T] or [_] icon on the status bar. You can also use the Ctrl+ Alt+ Shift+ E shortcut.

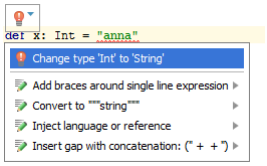


Enabling or Disabling Type-Aware Highlighting on the File Level

- Open the file and on the status bar click Hector icon. If you disable Hector, then the type-aware highlighting will be also disabled. Each time you enable or disable the type-aware highlighting, you can see the notification of the status's change in the Event log.

Working with Type-Aware Highlighting in Editor

- Let's see how it works:



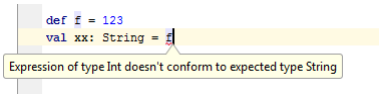
As you can see, the expression is wrong and the value is highlighted.

If the type-aware highlighting is disabled, the error is not highlighted and it might be quite difficult to spot:

```
def x: Int = "anna"
```

- You can also enable the highlighting based on expression type inference.

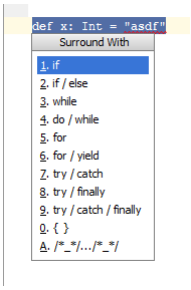
Consider the following example:



As the pop-up message suggests, the expression type is wrong and is highlighted.

Disabling Type-Aware Highlighting Locally

- Highlight the code and on the main menu select Code | Surround With.
- When Surround With window opens, select the /*_*/.../*_*/ option from the list.



After you've selected the appropriate option your code looks like this:

```
/*_*/def x: Int = "asdf"/*_*/
```

As a result, the error is not highlighted.

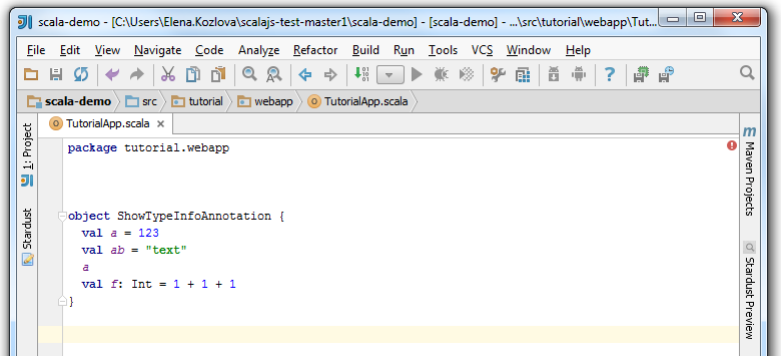
Working with Scala Show Type Info Action

IntelliJ IDEA offers an easy way to work with type inference feature such as Scala Show Type Info action.

- Entering code
- Invoking Show Type Info Action
- Adding Type Annotation
- Removing Type Annotation
- Seeing Type Annotation on Expression

Entering code

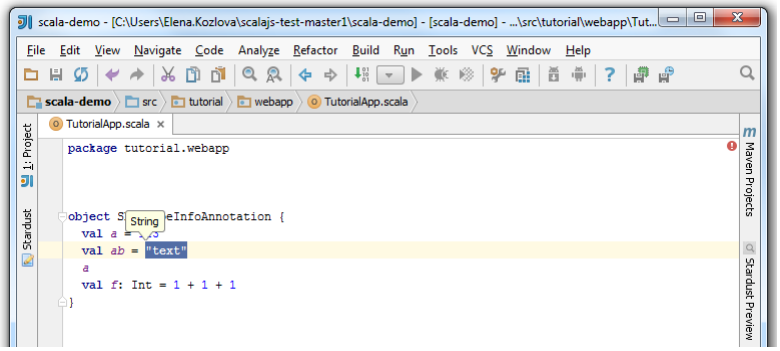
- Let's enter a random code and see what can be done:



Invoking Show Type Info Action

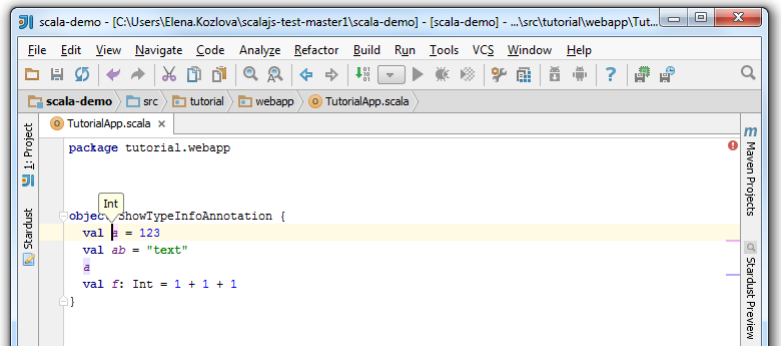
- If you select Show type info on mouse hover after, ms check box in Settings | Languages and Frameworks | Scala, then in the Editor, you can simply navigate with the mouse to a value to see its type information.
- To invoke Show Type Info action in Editor, navigate to the value and press Alt + =

  (Ctrl + Shift + P for OS X):



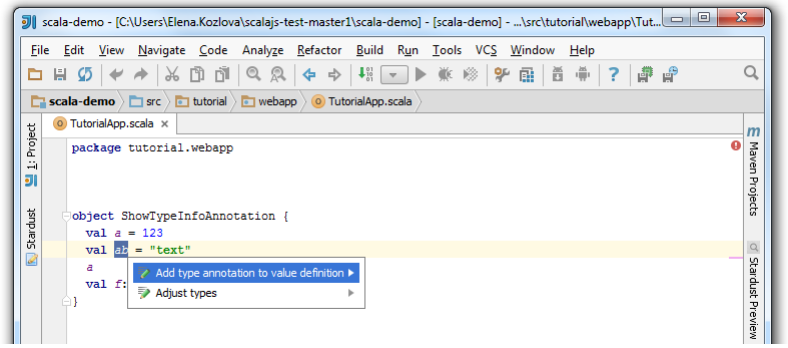
- You can also see the type information on a value definition. Simply put the caret on a value definition and press Alt + =

  (Ctrl + Shift + P for OS X):

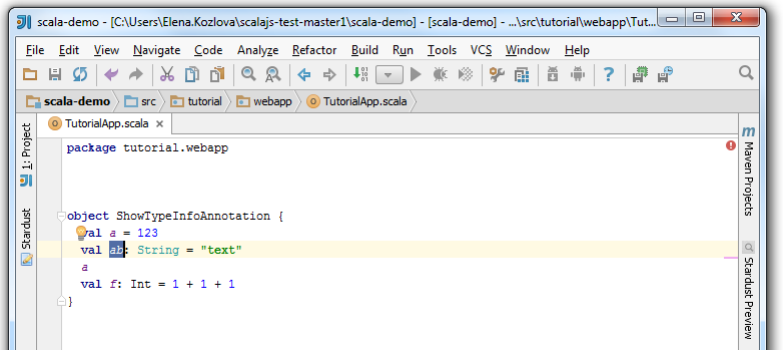


Adding Type Annotation

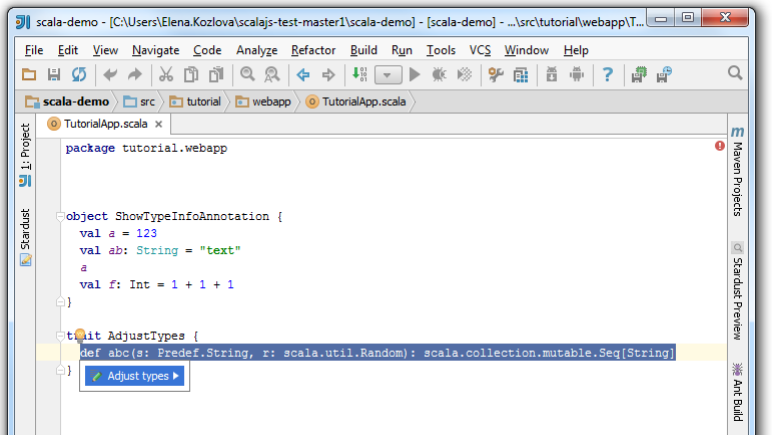
- To add a type annotation, highlight the value and press Alt + Enter:



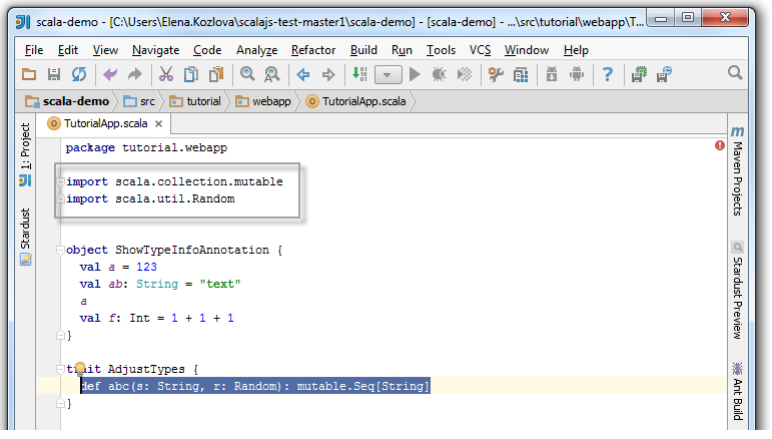
As a result, the type annotation is added:



– You can also use Adjust types action to shorten types with full qualified names.

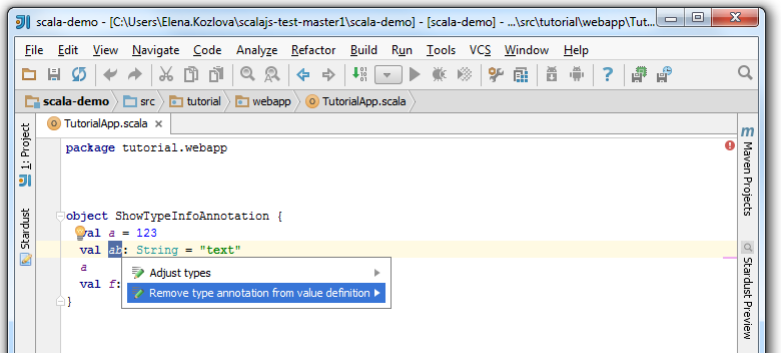


In this case the necessary imports are added.



Removing Type Annotation

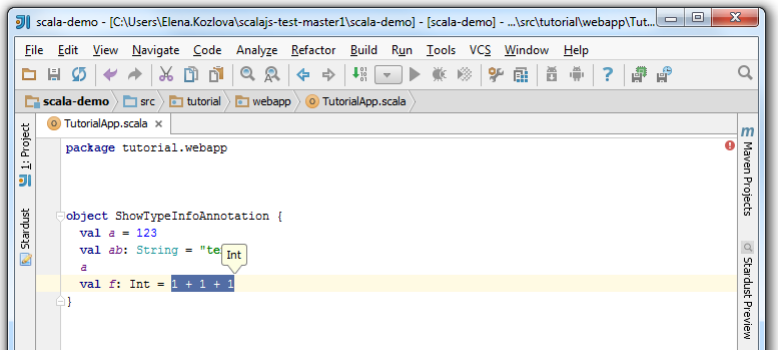
– You can also remove the type annotation the same way you have added it, by pressing Alt+ Enter:



Seeing Type Annotation on Expression

– To see the type information on an expression, simply navigate to the expression, highlight it and press Alt + =

(Ctrl + Shift + P for OS X):



Managing Imports in Scala

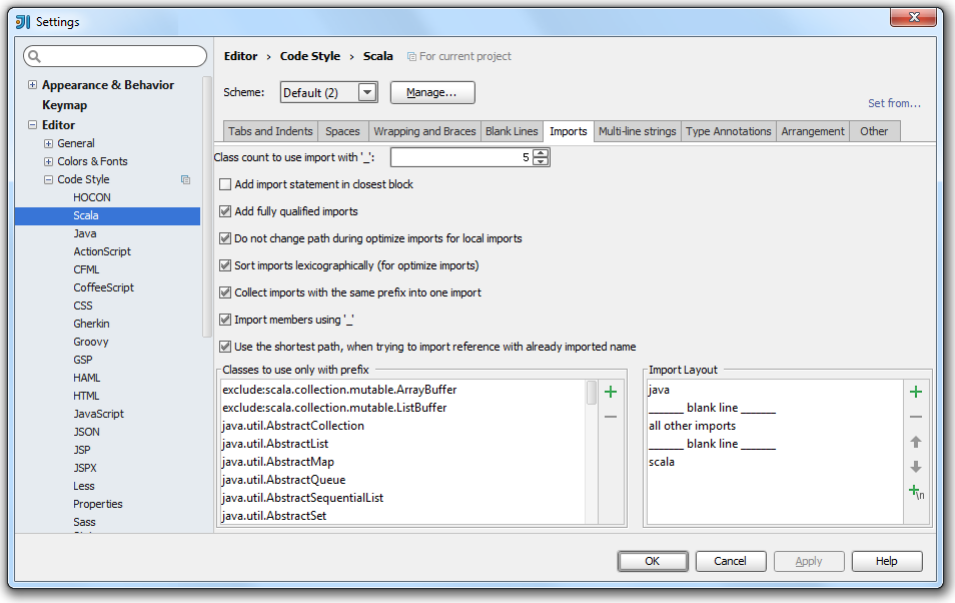
IntelliJ IDEA lets you manage your imports in Scala the same way as it does in other languages. You can configure imports in the Project Settings and in the IDE Settings. You can also optimize your imports and exclude classes from auto-import in the editor.

- Managing Scala Imports through the Scala Settings
- Managing Scala Imports through Auto Import Settings
- Optimizing Imports
- Excluding Classes from Auto-Import

Managing Scala Imports through the Scala Settings

You can format your imports using Project Settings.

1. Select File | Settings | Code Style | Scala.
2. On the Scala page, select Imports tab.

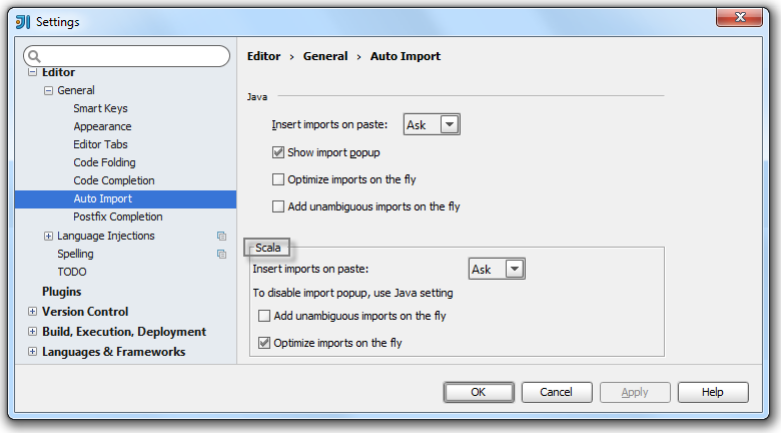


3. In the Imports tab, configure the Importing settings and press OK.

Managing Scala Imports through Auto Import Settings

You can configure the behavior of the imports in your workspace through the Auto Import settings.

1. Select File | Settings | Editor | General | Auto Import.



2. On the Auto Import page, in the Scala section, select from the following options:
   - Insert imports on paste - use this drop-down list to define how IntelliJ IDEA will insert imports for pasted blocks of code if they contain references to classes that are not imported into the target class.

     You can select from the following options:
     - All - select this option to have IntelliJ IDEA automatically add import statements for all classes that are found in the pasted block of code and are not imported in the current class yet.
     - Ask - if this option is selected, when pasting code blocks, IntelliJ IDEA will open a dialog box, where you can choose the desired imports.
     - None - select this option to suppress import.

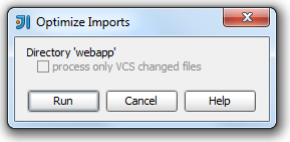
     If you skip an import suggested in the Ask mode or choose the None mode, the non-imported classes will be red-highlighted and an import pop-up window will appear to help you create import statements using the Alt + Enter keyboard shortcut.
   - Optimize imports on the fly - select this check box to have the Optimize Imports operation automatically performed for your files.

     If you clear this check box, you can manually optimize your imports selecting Code | Optimize Imports.

– Add unambiguous imports on the fly - select this check box to have IntelliJ IDEA automatically add imports that can be added without user intervention.

Optimizing Imports

You can optimize imports on the directory selecting Code | Optimize Imports command. In this dialog box, specify from where you want IntelliJ IDEA to remove unused import statements, in order to optimize the import procedure.



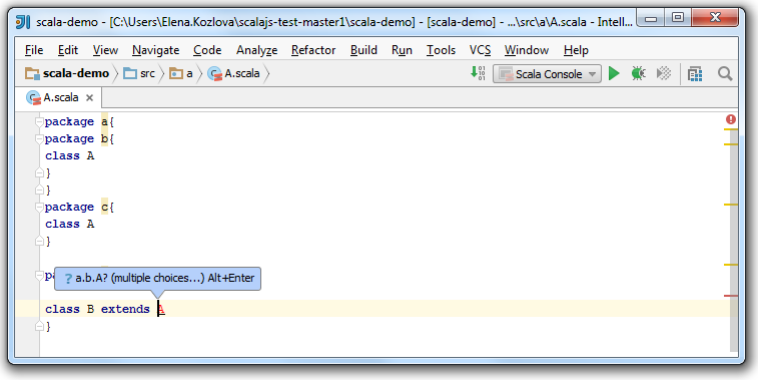
– Process only VCS changed files - if this check box is selected, then reformatting will apply only to the files that have been changed locally , but not yet checked in to the repository.

This check box is only available for the files under version control.

Excluding Classes from Auto-Import

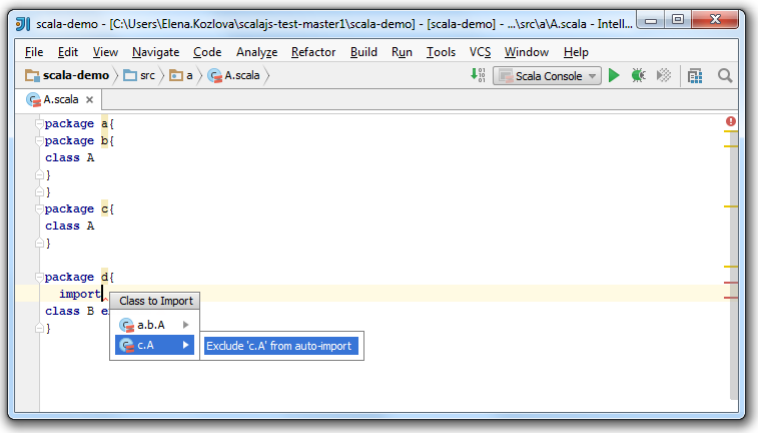
If the list of suggested imports is too wide, you can exclude unnecessary classes on the fly using intention actions.

1. Start typing a name in the editor.

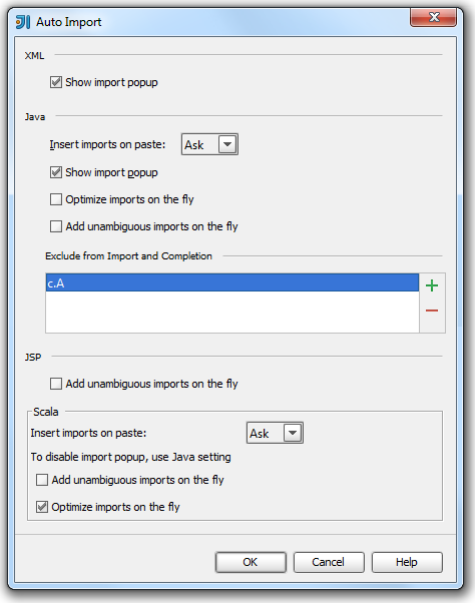


2. In the Class to Import suggestion list, click Alt + Enter, and click the right arrow to reveal the nested list of intention actions.

IntelliJ IDEA suggests you to exclude specific class or the whole containing package.



3. In the dialog that opens perform necessary changes and click OK.


Auto Import
XML
Show import popup
Java
Insert imports on paste:
Ask
Show import popup
Optimize imports on the fly
Add unambiguous imports on the fly
Exclude from Import and Completion
c.A
JSP
Add unambiguous imports on the fly
Scala
Insert imports on paste:
Ask
To disable import popup, use Java setting
Add unambiguous imports on the fly
Optimize imports on the fly
OK
Cancel
Help

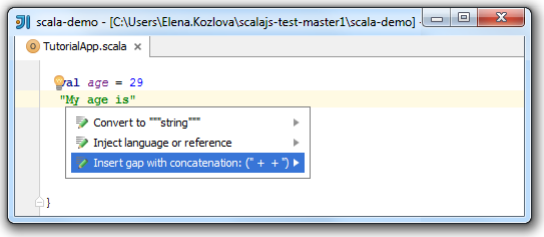
Working with Strings in Scala

IntelliJ IDEA lets you add different intentions for strings and perform different actions with strings in Scala. IntelliJ IDEA also supports the Scala multi-line strings feature. This feature lets you format literals and helps your code to look more orderly.

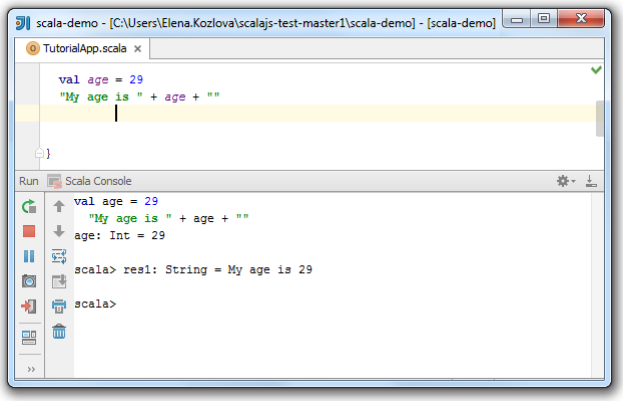
– Inserting Gap with Concatenation into a String
– Converting a String Concatenation into Interpolated String
– Converting a String into a Multi-line String and Vice Versa
– Inserting a Replace Intention
– Inserting an Inject Language/Reference Intention
– Editing Multi-line String Settings

Inserting Gap with Concatenation into a String

1. In your editor, enter a string and press Alt + Enter. The list of appropriate intentions opens.



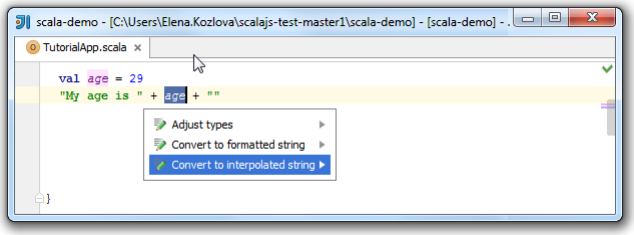
2. Select Insert gap with concatenation ("+ +") and press Enter. Now you can insert a value into your string. You can evaluate the result in the Scala Console window.



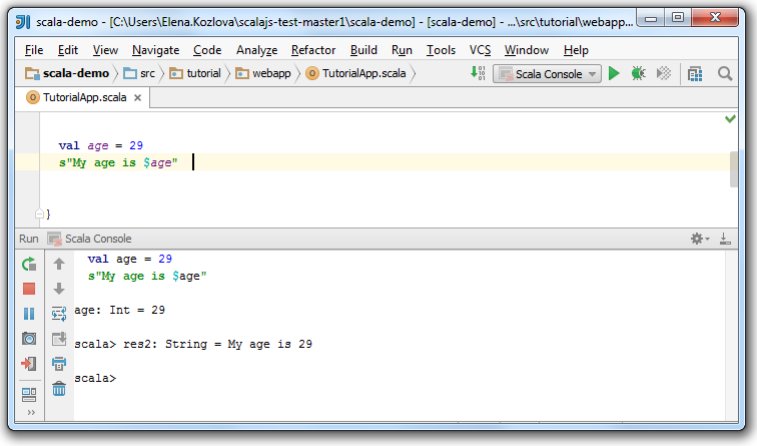
Converting a String Concatenation into Interpolated String

For the Scala version 2.10 and later you have the ability to convert a string concatenation into the interpolated string.

1. Select a value with concatenation in your string and press Alt + Enter. The list of appropriate intentions opens.



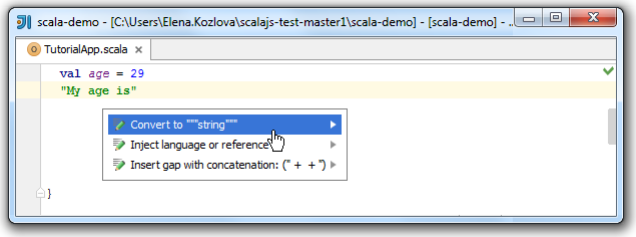
2. Select Convert to interpolated string, press Enter and view the result.



3. You can also choose an option Convert to formatted string and get your basic Java formatted string.

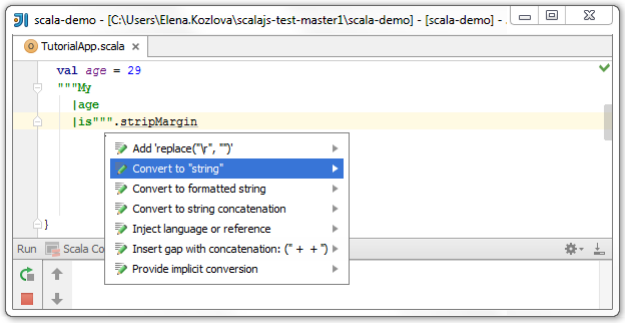
Converting a String into a Multi-line String and Vice Versa

- Enter your string, press Alt + Enter and select Convert to """string""" option in the drop-down list of intentions.

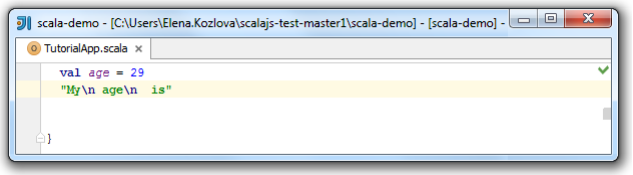


Converting to multi-line strings removes escaped sequences such as '\\' or '\n'.

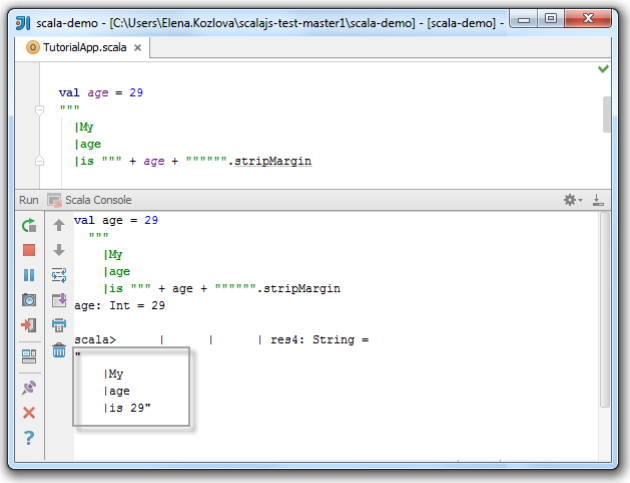
- You also can convert the multi-line string into the regular string. Press Alt + Enter to open the drop-down list of intentions. Select Convert to "string" and press Enter.



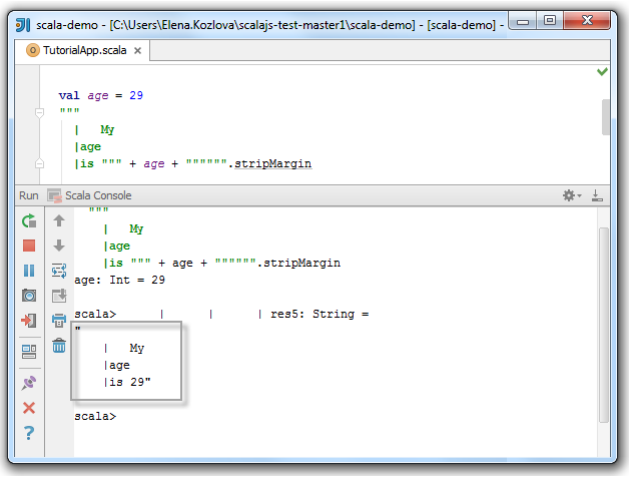
View the result.



- To start entering a multi-line string, simply type triple quotes in your editor. If you press Enter, it will automatically invoke stripMargin method. The stripMargin method removes the left-hand part of a multi-line string up to a specified delimiter.



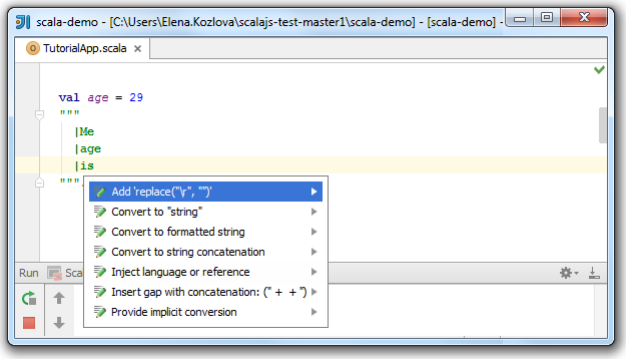
The white spaces are also preserved. Please see the following example:

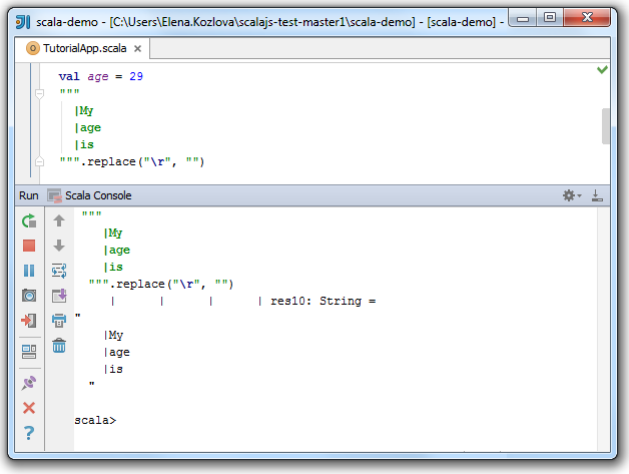


Inserting a Replace Intention

The replace("\r","") intention lets you keep the caret in the correct place on the next line in the multi-line strings regardless of what operating system you have at the moment.

− Enter a multi-line string, press Alt + Enter and select the appropriate intention from the drop-down list.

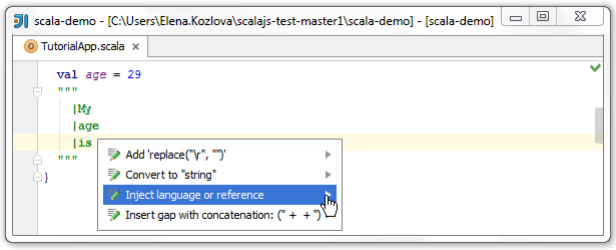




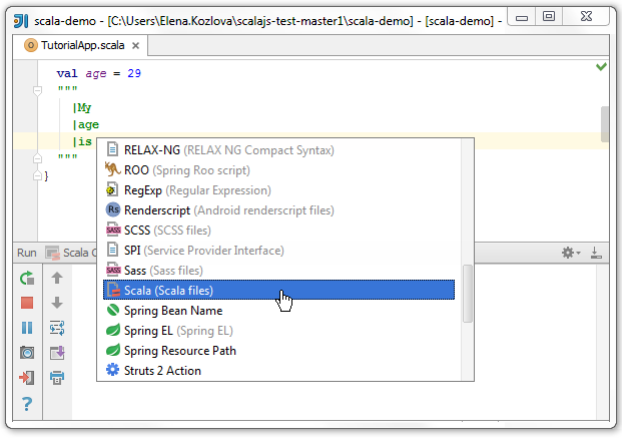
Inserting an Inject Language/Reference Intention

You can inject language or a reference into your multi-line string literals.

− Enter a multi-line string, press Alt+ Enter and select Inject Language/Reference intention from the drop-down list. The list of available languages opens.



− Choose the appropriate one and press Enter.



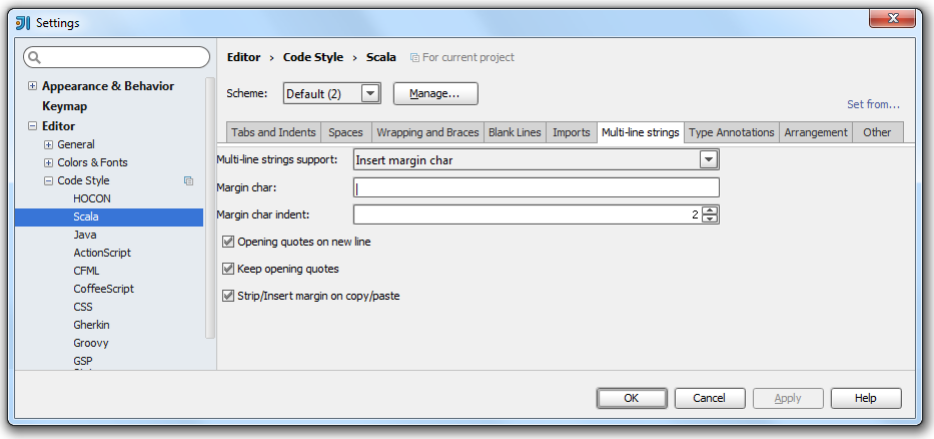
To cancel the language injection, simply choose the Un-Inject Language/Reference intention.

Editing Multi-line String Settings

You can set a different format for multi-line strings' options such as Margin char indent or Margin Char. You can also disable a multi-line strings support.

1. In the main menu, select File | Setting | Editor | Code Style | Scala.

2. On the Scala page, select the Multi-line strings tab.



Working with Scala Implicit Conversions

Scala implicit conversions is a powerful tool for Scala coding. It lets you decrease redundancy in your code and lets you make the code easier to read.

- Implicit Conversion Highlighting
- Invoking Implicit Conversion Methods
- Invoking Implicit Parameters

Implicit Conversion Highlighting

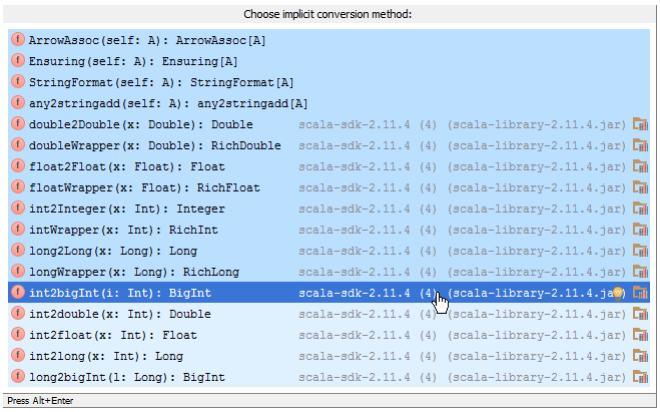
If Scala type-aware highlighting [T] is enabled then IntelliJ IDEA highlights methods that are added via implicit conversions with a grey underline:

```
val x: BigInt = 1
```

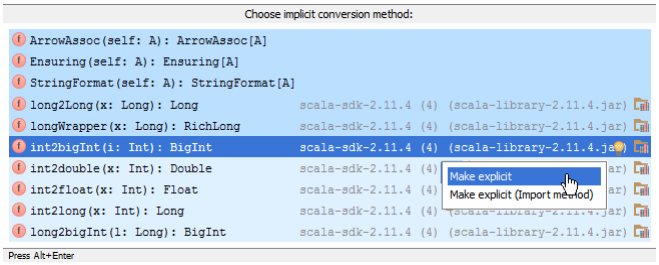
Invoking Implicit Conversion Methods

- Select an expression and press Ctrl + Shift + Q (Ctrl+Q for OS X) to invoke the list of applicable implicit conversions. The list shows the regular scope displayed on the top and the expanded scope that is displayed on the bottom of the list.

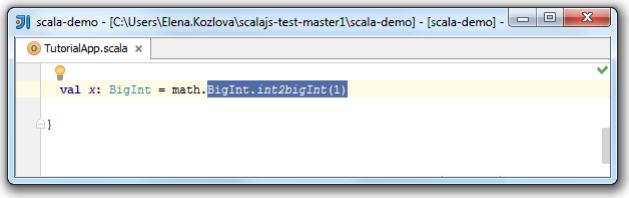
  Note Note that IntelliJ IDEA highlights an implicit conversion that was used for the selected expression. If IntelliJ IDEA cannot find the implicit conversion or if it finds more than one match then the list of Introduce Variable opens.



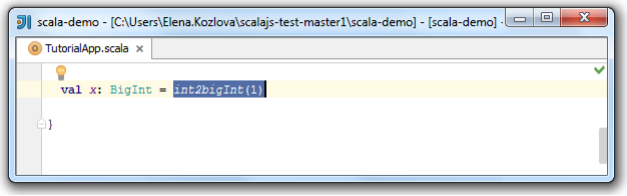
- You can make the implicit conversion method explicit. Press Alt + Enter and select Make explicit or Make explicit (Import method):



- If you select Make explicit then IntelliJ IDEA returns a method call with the class name. It might be helpful if you need to make sure that the compiler imports the particular implicit conversion method that you originally wanted:



- If you select Make explicit (Import method) then the method is imported statically and IntelliJ IDEA returns just its call without the class name. Also, the next time you open the list of useful implicit conversions you will see this method in the regular scope:

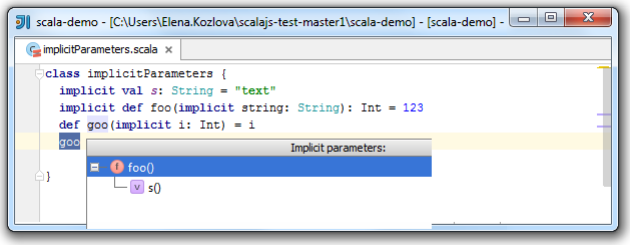


Invoking Implicit Parameters

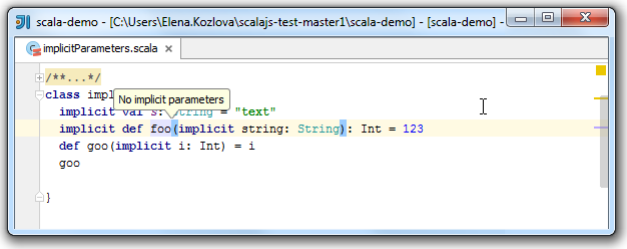
- Place a cursor to the method where implicit conversion was used and press Ctrl + Shift + P

  (Meta + Shift + P for OS X) to invoke implicit parameters. It might be helpful for code analyzing when you want to find out what implicit parameters were passed to the particular call. IntelliJ IDEA also lets you view the recursive implicit parameters.

  Note Note that IntelliJ IDEA highlights the method call where implicit parameters were used.



– If IntelliJ IDEA cannot find method calls where implicit parameters were passed, it displays a pop-up message:



Seam

In this section:

- Seam
  - Introduction
  - Developing applications using Jboss Seam
- Configuring Modules with Seam Support
- Defining Seam Components
- Viewing Seam Components
- Defining Seam Navigation Rules
- Defining Pageflow
- Navigating Within a Conversation
- Navigating Between an Observer and an Event

Introduction

Seam support in IntelliJ IDEA enables you to develop, run, debug and deploy applications using the facilities provided by this framework, by means of a dedicated Seam facet. Usually, Seam facet is used in modules in conjunction with the other facets required by the nature of your applications (for example, Web, EJB, JSF, Java EE, or Hibernate). Seam in IntelliJ IDEA make it easy to implement transactional models, Java Persistence API, JSF, and jPDL technologies.

Seam support involves:

- Automatic detection of Seam components.
- Detecting and visualizing Seam component dependencies.
- Possibility to detect and download the missing libraries and archives.
- Seam annotations.
- Injected expression language.
- Coding assistance (code completion, intention actions and quick fixes), which applies to the Seam annotations, components, navigation and pageflow definition files, and to injected expressions.
- Navigation, search and refactoring.

Besides the above mentioned features, IntelliJ IDEA supports business processes and pageflows provided by Seam:

- Possibility to define the page navigation rules, using the text editor, or the Navigations Graph.
- Possibility to define jPDL pageflow, using the text editor, or the Designer.

Developing applications using Jboss Seam

To develop an application using Jboss Seam, follow these general steps

1. Configure Seam support in your module.
2. Set up the required data sources, see Managing Data Sources .
3. Populate your application with the necessary classes.
4. Define the Seam components , using the `components.xml` file and Seam annotations.
5. Annotate the classes, using the other annotations (EJB, persistence etc.)
6. Define the page navigation rules .
7. Define the pageflow .

Tip In an application with Seam facet, you can:

- In addition to the regular means of navigation , you can jump between the parties of a conversation , or from an observer to an event .
- Explore Seam components dependencies .

Configuring Modules with Seam Support

Seam support requires a thorough approach to the module configuration. It is essential that all the necessary facets are enabled, and the libraries are defined. Thus you will be able to annotate your code, use import assistance, code completion and other productivity features of IntelliJ IDEA.

To configure Seam support in a module

1. Add Seam facet to the desired module.
2. If you have not configured libraries for Seam in advance, the facet page displays the list of missing libraries. To resolve the problem, click the Fix button. In the Specify Libraries dialog box, specify whether you would like to use one of the existing libraries, or download the missing archive. You can control the target location where the archive will be placed, the library name, and the level on which the library will be created.
3. Edit the configuration files provided by other facets (`web.xml` , `ejb-jar.xml` , `faces-config.xml` , hibernate configuration files etc) to include Seam-specific elements.
4. Create the necessary Seam files (`seam.properties` , `components.xml` , `pages.xml` etc.)

Defining Seam Components

You can define Seam components in two ways:

- By adding annotations to a class.
- By declaring a component in the `components.xml` file.

A component can be declared both with an annotation, and with an entry in the `components.xml` file. The `components.xml` file is used to specify additional properties of a component.

To annotate a class as a Seam component

1. Open the desired class for editing.
2. Add the `@Name` Seam annotation:

> Tip If the corresponding import statements are missing, use the import assistant prompt.

To create the components.XML file

1. In the Project tool window, right-click the directory where you want to create the `components.xml` file, for example, `WEB-INF` directory, and choose New | Seam components.xml on the context menu.
2. Populate the file as required. Note that Seam-aware coding assistance is available.

Viewing Seam Components

IntelliJ IDEA provides the possibility to view all Seam facets detected in your project in the Seam tool window . This view displays the list of all Seam components grouped by modules. The appropriate library components are also included.

For each module with a Seam facet, you can explore its structure using a dedicated dependencies graph. To open this graph, use the Show Seam Components Dependencies command available in the context menu for a module.

The read-only dependencies graph opens on a separate editor tab. You can change the way the graph is shown (zoom in and out, show or hide grid, change the layout, etc.) and perform other, basic operations.

Viewing Seam components and dependencies

1. To open the Seam tool window , choose View | Tool Windows | Seam. The modules with a Seam facet are shown.

   Tip You can navigate from a component to its definition in the Project view by pressing F4 .

2. Use the toolbar buttons or the keyboard shortcuts to expand (, Ctrl+NumPad Plus) or collapse (, Ctrl+NumPad -) the nodes.

3. To open the dependency diagram for a module, right-click the module of interest and select Show Seam Components Dependencies from the context menu.

4. To view or edit the source code for a component, open the component in the editor. To do that, select the component of interest and press F4 .
   To open the components which are the "leaves" of the tree (i. e. the ones at the bottom of the hierarchy), you can also use a double click.

Defining Seam Navigation Rules

On this page:

- Introduction
- Creating pages.xml file
- Opening page navigation rules file

Introduction

If you have to define views for a whole application, use the global `pages.xml` file. This file contains description of the navigation rules, page actions, and exception handling. For a large application this file can grow too large, and in this case it makes sense to create a number of fine-grained `*.page.xml` files, one file per page. So doing, the global items should be still described in the `pages.xml` file.

IntelliJ IDEA provides Navigation Graph as an additional tab to the textual source of the xml file. In this view you can explore the page navigation rules in a convenient way.

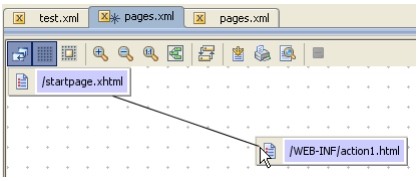
The Navigation Graph tab is only available for the global `pages.xml` file. All the `*.page.xml` files detected in a module are collected and included in the common graph.

It is very important to properly configure the Web resource directories for a Web facet of your module. In particular, the directories that contain pages to be included in the navigation rules, should be added to the list of Web resources.

Creating pages.XML file

To create pages.XML file

1. In the Project view, right-click the `WEB-INF` directory.
2. On the context menu, choose New | Seam pages.xml. Stub `pages.xml` file is created.
3. Populate the file with the necessary pages, and define navigation rules. You can do it in one of the two possible ways:
   - Open the Navigation Graph tab, in the Project view select the desired pages, and drag-and-drop them to the diagram background. Draw links between the nodes.



   - Open the Text tab, and type the source code for the navigation rules. Note that code completion is available for the paths to the pages, xml elements and expression language:

Note that source code of the page navigation file is fully synchronized with the Navigation Graph. Adding or removing a page or a link to/from the Navigation Graph is immediately reflected in the source code.

Opening page navigation rules file

To open the page navigation rules file

- In the Project view of the Project tool window, right-click the `pages.xml` file, and choose Jump to Source, or just press F4.

Defining Pageflow

On this page:

- Introduction
- Creating pageflow definition
- Actions, available in the Designer

Introduction

Enabling Seam facet in a module makes it possible to a create pageflow definition using jPDL. IntelliJ IDEA provides a dedicated graphic Designer, completely synchronized with the source code. In the Designer, you can create a stub pageflow, which is immediately reflected in the source code of the pageflow file. To make the pageflow definition usable, you have to additionally edit the source code of the pageflow file.

It is very important to properly configure the Web resource directories for a Web facet of your module. In particular, the directories that contain pages to be included in the pageflow, should be added to the list of Web resources.

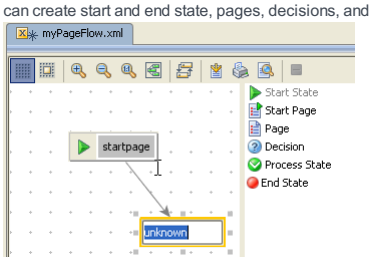
Creating pageflow definition

To create a pageflow definition

1. In the Project tool window, right-click the directory where you want to create the pageflow definition file, for example, `WEB-INF` directory, choose New | Seam pageflow on the context menu, and specify the file name. IntelliJ IDEA creates pageflow definition stub xml file, and opens it in a separate tab in the editor.

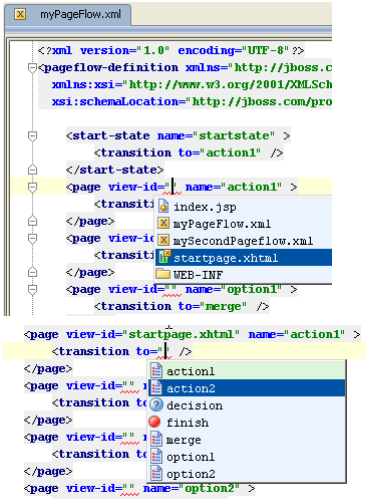
2. Click the Designer tab of the pageflow definition file.

3. Click the desired button in the toolbox of the Designer, and drag it to the Designer area. Add other nodes, and change their names as required. This way you can create start and end state, pages, decisions, and process states.



4. Draw links between nodes.

5. In the Text tab, edit the source code to associate it with specific views, add attributes and subelements (specifying action and decision expressions, transitions names, events, adding subprocesses to process states etc). Note the code completion for the attributes and values:



Actions, available in the Designer

The following actions are available in the Designer:

- Navigation: you can jump from the visual presentation of a pageflow in the Designer to the underlying source code. To do that, right-click an element, and choose Jump to Source on the context menu, or press F4.
- Show Usages and Find Usages: select a node, and press `Ctrl+Alt+F7` or `Alt+F7`.
- Rename refactoring: select a node, press `Shift+F6`, and specify the new name in the Rename dialog box. So doing, the corresponding reference in the underlying xml file is also updated. Note that if you rename a node using the in-place editor, its reference in the source code is not changed, and an error is highlighted.
- Deleting nodes and links: when a node or link is deleted from the graph, the corresponding element is deleted from the underlying xml file.

Navigating Within a Conversation

Seam support in IntelliJ IDEA includes support for conversations. The beginning and the end of a conversation are denoted with the `@Begin` and `@End` annotations respectively. In the source code editor, these annotations are marked with the gutter icons:

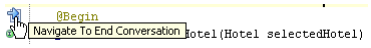
– jump to the beginning of a conversation.

– jump to the end of a conversation.

IntelliJ IDEA enables you navigate from the beginning of a conversation to its end, and vice versa.

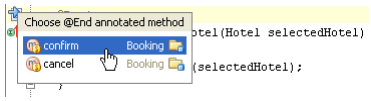
To navigate between the beginning and ending points of a conversation

– Click the gutter icon to the left from the @Begin annotation:



– Click the gutter icon to the left from the @End annotation:

If there are several possible end points of a conversation, you are prompted to select the desired one:

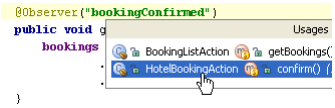


Navigating Between an Observer and an Event

If a method is marked as an observer of a certain event type, you can jump to a method where this event is raised, using the Show Usages command.

To navigate from an observer to an event to be observed

1. In the editor, place the caret at the event type value in the @Observer annotation.
2. Press `Ctrl+Alt+F7`, or choose Edit | Find | Show Usages on the main menu.
3. From the suggestion list, select the desired destination.



The class containing the method that raises the event in question, is opened in a separate tab, the caret resting at the event type value:

```
events.raiseTransactionSuccessEvent("bookingConfirmed");
```

Tip If an event is defined in the `components.xml` file, same kind of navigation is also available for it.

Spring

In this section:

- Spring
  - Spring support overview
  - Developing an application using Spring
- Enabling Spring Support
- Managing Spring Configuration Files
- Managing File Sets
- Using Spring Bean Patterns
- Using Spring Bean Templates
- Spring Boot

Spring support overview

To support the Spring framework , IntelliJ IDEA provides a set of plugins (the Spring Support plugin and others) and a dedicated facet type (Spring).

All Spring plugins are bundled with the IDE and are enabled by default.

The Spring facet is often used in combination with Hibernate .

The Spring support in IntelliJ IDEA includes:

- Complete coding assistance, including smart completion in configuration files.
- Spring-aware refactorings .
- Highlighting, inspections and quick-fixes .
- Visual Spring diagrams for viewing dependencies.
- A dedicated Spring file set editor.
- Built-in File Templates for Spring context files.
- Live Templates for beans and patterns.
- Spring aspect-oriented programming (AOP) support, with full coding assistance:
  - Smart code completion and highlighting.
  - AspectJ-aware code completion with syntax and error highlighting.
  - Numerous AspectJ-aware quick-fixes.
  - Enhanced navigation with active gutter icons.
  - Spring AOP inspections.
- The p-namespace support for properties definition.
- Spring annotations support.
- Support for filters to customize scanning.
- Parsing custom Spring beans
- Navigation for beans defined with Spring custom namespaces.
- Spring 2.5: `bean()` pointcut designator support.
- Find by symbol.
- Quick Javadoc (`Ctrl+Q`) on bean references.
- Coding assistance for `@Configuration` -annotated Java classes and related annotations (`@Bean` , `@Import` and `@DependsOn`).
- Quick navigation from `@ContextConfiguration` -annotated JUnit test cases to corresponding Spring configuration files by means of active gutter icons (see the screencast).
- Support for certain Spring 3.1 features including:
  - The c-namespace support with code completion , error highlighting, quick fixes and code generation .
  - Bean definition profiles support. In addition to code completion and error highlighting, the following features are available:
    - For xml files containing profile definitions (`<beans profile=...`), an additional status bar is provided at the top of the editor. Shown in this bar are the names of the profiles which are currently active. This bar also accommodates the controls for activating and deactivating the profiles.
    - Definitions of active and inactive profiles are shown in the editor in different colors.
    - Quick fixes (for found problems) are suggested only for the definitions that are currently active.
    - You can search for profiles in the xml configuration files, `@Profile` annotations and Java code and, if necessary, perform the Rename refactoring for them.
- Spring Security support:
  - Almost all tags are supported; new Spring Security 3.1 namespace elements (e.g. `<jee>` , `<session-management>` , etc.) are supported in full. This includes code completion , navigation and the rename refactoring .
  - Language injections for AOP and SQL are available.
  - Spring Security OAuth 1.0 and 2.0 namespaces are supported.

  See also, Spring Security Support in IntelliJ IDEA 12 .
- Spring Integration support (up to version 2.2):
  - All existing namespaces are supported (including `AMQP` , `Gemfire` , `JPA` and `Redis`). Coding assistance includes code auto-completion , highlighting, quick-fixes , inspections , and navigation.
    The `rabbit` namespace is supported out-of-the-box, simplifying the integration with AMQP.
  - Language injections are available for all attributes containing Spring EL expressions (including references to header names), `<groovy:script>` and everything containing `XPath` expressions.
  - The rename refactoring is supported throughout the framework.
  - The Spring dependencies diagram lets you view the element dependencies and navigate to the referenced beans.

  See also, Spring Integration in IntelliJ IDEA 12 .

- Spring Batch support with the ability to download and configure the Spring Batch libraries. Coding assistance features include code completion and highlighting in XML files, and the rename refactoring . Predefined Spring EL variables and references to step-scoped beans are resolved automatically. For more information, see Support for Spring Batch in IntelliJ IDEA 12 .
- Spring Web Flow support:
  - Structure view is available for `flow.xml` .
  - Spring EL is fully supported in `flow.xml` . This includes code completion , navigation and refactoring .
    All built-in and custom attributes are available on JSP pages.
  - The Navigate | Symbol option (`Ctrl+Shift+Alt+N`) lets you quickly navigate to any of the states. In the editor, gutter icons are provided for navigating to parent or child elements. To access a parent or child flow or subflow, you can use Navigate | Related Symbol (`Ctrl+Alt+Home`).
  - All Spring Web MVC views are accessible.

  See also, Spring Web Flow Support in IntelliJ IDEA 12 .
- Spring Data JPA support. See Spring Data JPA in IntelliJ IDEA 11 .
- Support of the spring-websocket module.
- Support for Spring Boot projects and integration of Spring Initializr into the New Project wizard. For more details, see Spring Boot .

When using Spring and Hibernate in a project simultaneously, the following integration features are available:

- Dedicated templates to support Hibernate in Spring configuration files.
- Hibernate-aware code completion in Spring configuration files.

- The support of AbstractAnnotationConfigDispatcherServletInitializer in Spring MVC . It might be useful for Spring MVC configurations.

Developing an application using Spring

Before you start developing a Spring application, enable the Spring support in IntelliJ IDEA .

The tasks specific to Spring application development are discussed in the following topics:

- Managing Spring Configuration Files
- Managing File Sets
- Using Spring Bean Templates
- Using Spring Bean Patterns

For general instructions, refer to How-to .

Enabling Spring Support

To enable Spring support:

- Make sure that all the necessary IntelliJ IDEA plugins are enabled. These are the Spring Support plugin (for general Spring support) and also a number of other, more specific plugins (Spring Security, Spring Integration, etc.).
- Create a Java module with a Spring facet or add a Spring facet to an existing Java module.

On this page:

- Making sure that the Spring Support and other Spring plugins are enabled
- Creating a module with a Spring facet
- Adding a Spring facet to an existing module

To make sure that the Spring Support and other Spring plugins are enabled

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left pane, select Plugins.
3. In the right pane, on the Plugins page , type `spring` in the search box. As a result, only the plugins whose names and descriptions contain `spring` are shown in the list of plugins.
4. If the check box to the left of Spring Support is not selected, select it. If you are going to use particular Spring frameworks (Spring Security, Spring Integration, etc.), make sure that the check boxes next to their names are also selected.
5. Click OK in the Settings dialog.
6. If suggested, restart IntelliJ IDEA.

To create a module with a Spring facet

Depending on the situation, you can choose to create a new project or add a module to an existing project.

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Spring . In the right-hand part of the page, specify the JDK that you are going to use.
3. Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download and use a Spring implementation library.
     To do that, select Download.

     Now, to view or modify the associated options, click Configure, and in the Downloading Options dialog that opens:
     - Select the Spring version that the library should implement.
     - Specify the library name.
     - Select the library level (global, project, or module).
     - Under Files to download, select which of the files you want to download.
     - Under Copy downloaded files to, specify the path to the destination folder. If you want to change the default path, click and specify the folder location in the dialog that opens .
   - Use a Spring library IntelliJ IDEA is already aware of.
     To do that, click Use library and select the required library from the list.

     If necessary, configure the library settings (for example, change its name). This is done in the Edit Library dialog which you can open by clicking Configure.
   - Create a new library using the appropriate JAR files available on your computer.
     To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (Keep the `Ctrl` key pressed to perform multiple selection.)

     If necessary, configure the new library (for example, change its name or level). To do that, click Configure and specify the required settings in the Create Library dialog.
   - Postpone setting up the library until a later time. In this case, select Set up library later.
4. If you want to enable support for more specific Spring features or frameworks (for example, Spring Security, Spring Integration, etc.), select the corresponding check boxes and specify the associated library options. The procedure is similar to that for the Spring library.
   Click Next.
5. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

To add a Spring facet to an existing module

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the module and select Add Framework Support.
3. In the left-hand pane of the Add Frameworks Support dialog, select the Spring check box.
4. Select the required library option and, if necessary, specify the associated settings. You can choose to:
   - Download and use a Spring implementation library.

To do that, select Download.

Now, to view or modify the associated options, click Configure, and in the Downloading Options dialog that opens:

- Select the Spring version that the library should implement.
- Specify the library name.
- Select the library level (global, project, or module).
- Under Files to download, select which of the files you want to download.
- Under Copy downloaded files to, specify the path to the destination folder. If you want to change the default path, click … and specify the folder location in the dialog that opens .

- Use a Spring library IntelliJ IDEA is already aware of.
  To do that, click Use library and select the required library from the list.

  If necessary, configure the library settings (for example, change its name). This is done in the Edit Library dialog which you can open by clicking Configure.
- Create a new library using the appropriate JAR files available on your computer.
  To do that, click Use library and then click Create. Select the required JAR files in the dialog that opens . (Keep the Ctrl key pressed to perform multiple selection.)

  If necessary, configure the new library (for example, change its name or level). To do that, click Configure and specify the required settings in the Create Library dialog.
- Postpone setting up the library until a later time. In this case, select Set up library later.

5. If you want to enable support for more specific Spring features or frameworks (for example, Spring Security, Spring Integration, etc.), select the corresponding check boxes and specify the associated library options. The procedure is similar to that for the Spring library.

6. Click OK in the Add Frameworks Support dialog.

Managing Spring Configuration Files

Spring configuration files contain bean definitions and other information. They are used to provide context information to the Spring framework.

IntelliJ IDEA lets you manage "traditional" XML configuration files and also `@Configuration`-annotated Java classes.

Mutually-related configuration files can be arranged in file sets to make their editing more effective.

When working with configuration files in the editor, you can generate beans, properties, dependencies, etc. using the IntelliJ IDEA's code generation feature (Code | Generate or `Alt+Insert`). You can also analyze the structure of your application using the Spring dependencies diagram .

- Creating a New XML Configuration File
- Adding an XML Configuration File to a File Set
- Creating a @Configuration-annotated Java Class

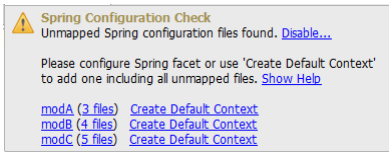
Creating a New XML Configuration File

1. In the Project tool window , select the package or directory in which you want to create a configuration file. Then choose File | New or press `Alt+Insert` .
   Alternatively, right-click the package or directory, and click New.
2. Select XML Configuration File and then select Spring Config.
3. In the New File dialog, specify the name of the configuration file and click OK.
   As a result, the new configuration file is created and opened in the editor.
4. If necessary, add the configuration file to a file set .

Adding an XML Configuration File to a File Set

There are three ways to add an XML configuration file to a file set:

- Unless an XML configuration file is included in at least one file set , the Configure File Set link is available in the editor for the configuration file. This link is located above the editing area, in the upper-right corner of the corresponding editor tab.
  1. Click the Configure File Set link.
  2. If you haven't defined any XML file sets yet, the Edit Application Context dialog opens. Configure the file set as required and click OK.
     If at least one XML file set already exists, you can add the configuration file to an existing file set or create a new file set and add the configuration file to it. To do that, select the necessary option from the Choose file set menu:
     - To add the configuration file to an existing file set, select the necessary file set.
     - To add the configuration file to a file set that does not yet exist, select <new file set>. Then, in the Edit Application Context dialog , configure the file set and click OK.
- Click the Create Default Context link in the upper-right corner of the editor tab where the unmapped configuration file is opened:



- Add a configuration file to a file set by editing the relevant file set as described in Managing File Sets . See also Creating a file set.

Creating a @Configuration-annotated Java Class

The procedure for creating `@Configuration`-annotated Java classes is the same as for creating "ordinary" classes, see Creating Java Classes, Interfaces, Enumerations and Annotations .

Managing File Sets

Spring configuration file sets, or just file sets, for short, are a way of grouping Spring configuration files in IntelliJ IDEA. They are intended for making your work with mutually related configuration files more effective.

IntelliJ IDEA lets you create file sets that contain XML configuration files or `@Configuration`-annotated Java classes.

– About file sets
– Parent-child relationships between file sets
– Creating a file set
– Editing a file set
– Removing a file set

About file sets

By combining Spring configuration files in a file set you tell IntelliJ IDEA that these files are mutually related. In this way you form a common editing context for these configuration files and get all the associated coding assistance.

To illustrate, let us consider an imaginary XML configuration file `config-one.xml` containing two bean definitions:

```
...
                    <bean id="beanOne"
                    class="BeanOne">
                    ...
                    </bean>
                    <bean id="beanTwo"
                    class="BeanTwo">
                    <constructor-arg ref="beanOne"/>
                    ...
                    </bean>
                    ...
```

As you can see, the second bean (`beanTwo`) references the first one (`beanOne`). Obviously, there is no problem with this reference while both bean definitions are in the same file.

Now let us assume that you want to move the definition of `beanTwo` to a separate configuration file, say, `config-two.xml` . In this case, you should somehow indicate that `ref="beanOne"` in `config-two.xml` is a reference to `beanOne` defined in `config-one.xml` . In IntelliJ IDEA, you can do that by combining `config-one.xml` and `config-two.xml` in a file set.

In a similar way, you can form file sets that contain `@Configuration`-annotated Java classes.

You can create as many file sets as you need; any configuration file may be included in more than one file set. The configuration files that are external to your project can also be included in your file sets.

On the other hand, you don't have to create file sets if you don't need them.

In certain cases, IntelliJ IDEA can create file sets automatically. This is the case for Spring Web applications when the corresponding configuration files are specified in `web.xml` or have recognizable file names (for example, `<servlet-name>-servlet.xml`).

Parent-child relationships between file sets

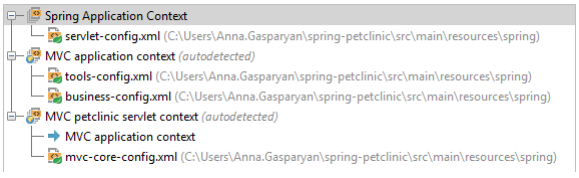
For each file set that you create, you can specify a parent file set. This lets you build file set hierarchies and, thus, control bean visibility at the stage of developing your application.

This IntelliJ IDEA feature fully mirrors the concept of hierarchical application contexts existing in the Spring framework where it's possible to use common beans from parent contexts, and inherit and overload the beans in child contexts.

The beans defined in a parent file set (or, to be more exact, in one of the configuration files belonging to a parent file set) are all visible in its child file sets (and all the descendent file sets). The beans defined in a child file set, however, are not visible in its parent file set.

Going back to the example we have discussed earlier , you could place `config-two.xml` in a file set which is a child of a file set containing `config-one.xml` . In such a case, the reference `ref="beanOne"` would still be valid (as a reference to a bean defined in a parent file set). On the other hand, the beans in `config-one.xml` could not reference the beans defined in `config-two.xml` .

In the user interface of IntelliJ IDEA, parent file sets have a blue arrow next to them ( ).



To create a file set

1. Open the Spring facet settings .
2. On the Spring Facet page , click + (Alt+Insert) and select one of the following options:
   – XML. Select this option to create a file set containing XML configuration files.

– Annotation Config. Select this option to create a file set containing `@Configuration`-annotated Java classes.

3. In the Edit Application Context dialog :
   1. Specify the name of the file set in the Name field.
   2. If necessary, select a parent file set from the Parent fileset list.
   3. Select the configuration files to be included in the file set by selecting the corresponding check boxes.
   4. If you want to add a configuration file external to your module or project, click Locate File and select the necessary configuration file in the Select Path dialog .
   5. Click OK.
4. Click OK in the Project Structure dialog.

You can initiate creation of a file set in the editor, if an XML configuration file which is not included in any of the file sets is currently open (see).

To edit a file set

1. Open the Spring facet settings .
2. If you just want to remove unnecessary configuration files from a file set, select the corresponding configuration files, click (`Alt+Delete`), and then click OK. Note that this does not delete the configuration files physically but just removes them from the corresponding file set.
   Otherwise, select the necessary file set and click (`Enter`).
3. Make the necessary changes in the Edit Application Context dialog .
4. Click OK in the Project Structure dialog.

To remove a file set

1. Open the Spring facet settings .
2. Select the file set to be removed, click (`Alt+Delete`), and then click OK. Note that this operation does not delete the corresponding configuration files.

Using Spring Bean Patterns

When developing application using Spring framework, you'll most likely need to use some persistence, for example, Hibernate. To bind a data access to Spring configuration, you need to create corresponding beans. To save your time, IntelliJ IDEA provides Bean patterns, so you don't have to manually code those beans. These patterns include beans for:

- Hibernate
- JDO
- Toplink
- iBatis
- JPA

There are also bean patterns for SDK Timer and OpenSymphony Scheduler.

Because all procedures are similar, we'll describe how to use Hibernate pattern, for instance.

To create Spring beans for compatibility with Hibernate

1. Place the caret within the Spring configuration file, and press `Alt+Insert`.
2. In the Generate menu select Patterns | Hibernate.
3. In the Add Hibernate Beans dialog box, select what beans you want to create. By default, required ones are already selected. If Hibernate libraries are missing in the module, click Fix to configure them. Click OK.
4. IntelliJ IDEA guides you through creating beans. You need only to select desired values for bean attributes.

Using Spring Bean Templates

IntelliJ IDEA provides a rich set of Spring bean templates that enable quick creation of various kinds of beans.

To add a template bean

1. Place the caret within the Spring configuration file, and press `Alt+Insert`.
2. In the Generate menu, select Template Beans | <template-group> | <template-name>. For example, Template Beans | Common Beans | Resource Factory Bean.
3. IntelliJ IDEA guides you through creating a bean. You need only to select the desired values for bean attributes.

Working with Spring Dependencies

IntelliJ IDEA lets you analyze your Spring project dependencies.

Opening Spring dependency graph

- In the Project View window right-click on the Spring configuration file for which you wish to create a diagram and select Diagrams | Show Diagram | Select Diagram Type.
- In the Select Diagram Type dialog select the diagram you want to create .
- Depending on the type of the diagram you have selected, IntelliJ IDEA opens the graph in the editor where you can analyze the relationship between the dependencies.

You can use the diagram toolbar and the context menu to check additional options and adjust your graph. You can also use the diagram settings to change the toolbar options.

Reading Spring Model Dependency Diagram

IntelliJ IDEA lets you view the relationships between Spring configuration files dependencies. IntelliJ IDEA uses different types of lines and colors to show how those relationships were created.

-

  A solid line and blue color are reserved for `@ComponentScan` annotation dependencies.

-

  A solid line and orange color are used for `@Enable` annotation dependencies.

-

  Dashed lines and green color are reserved for `Import` annotations.

-

  Lines with red color are used for errors.

  If some configuration file is included several times then all its inclusions except one are considered errors.

-

  Dotted lines are used for other dependencies.

Reading Spring Bean Dependency Diagram

IntelliJ IDEA lets you view the relationships between Spring bean dependencies. IntelliJ IDEA also uses different types of lines and colors to show how those relationships were created.

- A dashed line and blue color
- A dotted line and green color
- A solid line and grey color

Spring Boot

The Spring Boot support in IntelliJ IDEA includes:

- Integration of Spring Initializr that allows creating a custom Spring Boot starter project using a pre-configured project template directly from the New Project wizard .
- Advanced coding assistance for editing your application configuration which includes code completion, error highlighting, navigation and quick fixes.
- A dedicated Run/Debug Configuration that lets you quickly override the Spring Boot settings.

In this section:

- Creating Spring Boot Projects
- Adding Spring Boot Modules to Existing Projects

Creating Spring Boot Projects

To create a Spring Boot project, you can use Spring "starters" via Maven / Gradle , or the Spring Initializr service. Spring Initializr is a web-based tool that allows you to generate a template project and automatically sets all required dependencies according to the selected technologies instead of configuring them manually. It generates a full project, or a build file with Maven or Gradle.

With the Spring Initializr integration in IntelliJ IDEA, you can create a ready-to-run application in just a few clicks directly from the New Project wizard . To do this, perform the following steps:

1. If no project is currently open, click Create New Project on the welcome screen; otherwise select File | New | Project.
2. On the first page the New Project wizard , select Spring Initializr in the left pane.
3. In the right pane, specify a project SDK (JDK). If the necessary JDK is already defined in IntelliJ IDEA, select it from the list.
   Otherwise, click New and select JDK from the list. Then, in the dialog that opens , select the installation folder of the desired JDK (by this time, the corresponding JDK must already be installed on your computer).
4. Specify the Spring Initializr instance URL. By default, it is `https://start.spring.io` , but you can use any other custom instance if needed.
   Click Next to continue.
5. On the next page of the wizard, specify project metadata settings. Note that specific fields vary slightly depending on the Spring Initializr instance URL you specified in the previous step.
   Click Next to continue.
6. On the next page of the wizard, select the Spring Boot version and specify project dependencies.
   Click Next to continue.
7. On the last page of the wizard, check the project name and location and modify them if necessary.
   Click Finish to complete the wizard.

IntelliJ IDEA will create a fully-functional project with the predefined structure, a Spring facet , a Maven or Gradle build file and a set of libraries.

Note that if the Create run configuration for Spring Boot application option is enabled in Maven. Importing settings, IntelliJ IDEA will also automatically create a run/debug configuration for your application on Maven import.

Adding Spring Boot Modules to Existing Projects

You can either create a new Spring Boot module manually through Maven / Gradle , or use the Spring Initializr service.

To create a new Spring Boot module and add it to a project through IntelliJ IDEA, perform the following steps:

1. Open the Project Structure dialog and select Modules in the left pane.
2. Right click the project root and select Add | New Module from the context menu.
3. In the New Module wizard that opens, specify the Spring Initializr instance URL. By default, it is `https://start.spring.io` , but you can use any other custom instance if needed.
   Click Next to continue.
4. On the next page of the wizard, specify project metadata settings. Note that specific fields vary slightly depending on the Spring Initializr instance URL you specified in the previous step.
   Click Next to continue.
5. On the next page of the wizard, select the Spring Boot version and specify project dependencies.
   Click Next to continue.
6. On the last page of the wizard, enter the module name, check the content root and module file location and modify them if needed.
   Click Finish to complete the wizard.

Struts Framework

In this section:

- Struts Framework
  - Introduction
  - Developing Web applications with Struts
- Struts Data Sources
- Preparing to Use Struts
- Managing Struts Elements
- Managing Tiles
- Managing Validators
- Using the Web Flow Diagram

Introduction

IntelliJ IDEA supports the Struts framework for creating Java EE and Web Applications.

Struts support is enabled through the dedicated Web and Struts facets. The dedicated facets that contain settings, configuration file paths, and validation rules determine the structure of the module so that IntelliJ IDEA detects how to treat the module contents.

- A Struts facet can be only created as a child of a Web facet .
- Only one Struts facet is allowed per module.

IntelliJ IDEA provides the following support of the Struts framework:

- Automatic creation of the specific module structure and the Web application deployment descriptor `web.xml` based on the dedicated Web facet.
- Downloading and adding the necessary libraries automatically and creating the Struts application configuration file `struts-config.xml` based on the dedicated Struts facet .
- Coding assistance, including code completion, syntax and error highlighting, documentation lookup, and refactoring.
- Go to Symbol and support for Actions.
- Code inspections .
- Structure tool window with native Struts application structure navigation and Struts actions.
- A Struts Assistant tool window that provides facilities to manage Struts elements .

Developing Web applications with Struts

To develop a Web application using Struts, follow these general steps

1. Enable Struts support.
2. Create and edit the Struts elements and their interaction .
3. Create and edit tiles.
4. Create and edit validators.
5. Populate your application with the necessary classes.
6. Deploy and run your application.

Struts Data Sources

Struts Data Sources are managed in the Struts configuration file `struts-config.xml` .

IntelliJ IDEA provides a user-friendly visual interface for creating , editing , and removing data sources.

Data source elements are created and removed in the same way as other elements.

Editing a data source includes editing its properties and handling its attributes .

Preparing to Use Struts

On this page:

- Before You Start
- Basics
- Creating a Module with a Struts Facet
- Adding a Struts Facet to a Module
- Customizing the Struts Facet

Before You Start

Make sure the Web and Struts1.x bundled plugins are enabled. The plugins are bundled with IntelliJ IDEA and activated by default. If the plugins are not activated, enable them on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Basics

Integration with Struts is enabled through the Web and Struts1.x facets. These dedicated facets contain settings, configuration file paths, and validation rules. This information determines the structure of a module so IntelliJ IDEA detects how to treat the module contents. See Facet for details.

A Struts1.x facet can be only added as a child of a Web facet . Note that only one Struts1.x facet is allowed in a module.

Creating a Module with a Struts Facet

When you create a Struts module with the dedicated Web and Struts facets, IntelliJ IDEA configures the module, downloads and adds all the necessary libraries automatically.

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java . In the right-hand part of the page, specify the JDK that you are going to use.
3. Under Additional Libraries and Frameworks, select the Web Application check box.
   Select the version of the Servlet specification to be supported from the Versions list.

   If you want the deployment descriptor `web.xml` file to be created, select the Create web.xml check box.
4. Select the Struts check box.
5. Select the Struts version from the Version list.
6. You'll need a library that implements Struts. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the library files that implement the selected Struts version. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

   Click Next.
7. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

IntelliJ IDEA configures the new module as follows

- Creates the structure of the module with the Web and WEB-INF nodes.
- Creates a Web application deployment descriptor `Web.xml` under the WEB-INF node for versions 2.2 - 2.5.
- Creates a Struts configuration file `struts-config.xml` under in the WEB-INF node.
- Configures an Action servlet in `Web.xml` .
- Downloads and installs the libraries that implement Struts.

Adding a Struts Facet to a Module

When you add a Struts facet to an existing module, IntelliJ IDEA downloads and adds the necessary libraries automatically.

A Struts facet can be added only as a child of a Web facet. If the module has no Web facet, add it first. Also note that only one Struts facet is allowed in a module.

This will change the configuration of the module .

1. Open the Module Settings dialog box.
2. In the Modules node, right-click the relevant module.
3. To add a Web facet, choose New from the context menu. In the drop-down list of available facets, select Web.
4. Right-click the Web node and select New on its context menu.
5. Select Struts on the context menu of available facets. The right-hand pane displays the Facet 'Struts' dialog.
6. Customize the Struts facet settings in the Struts Features and Validation tabs.
7. If some additional libraries are required, the corresponding warning message displays. Click Fix. The necessary libraries will be downloaded and added automatically.

Customizing the Struts Facet

After you have enabled the basic Struts support, IntelliJ IDEA provides the ability to add more Struts features by customizing the Struts dedicated facet.

1. Open the settings of the Struts facet from the module in question .
2. The Struts Features tab shows a list of additional Struts components:
   - Struts taglib
   - Struts EL-taglib
   - Tiles
   - Validator
   - Struts-Faces
   - Scripting
   - Extras

   To enable support of a component from the list, select the check box next to it.
3. In the Validation tab, specify the files to validate and the validation settings:
   - To enable validating the configuration file of a particular component, select the relevant check box. IntelliJ IDEA will check whether the selected component can be supported correctly using the available resources and suggest to download the missing libraries, if necessary.
     Validation can be enabled for the configuration files of the following components:
     - Struts
     - Tiles
     - Validator
   - Specify the validation settings:
     - To get informed on errors detected during validation, select the Report errors as warnings check box.
     - To exclude Struts property keys from validation, select the Disable property keys validation check box.

Managing Struts Elements

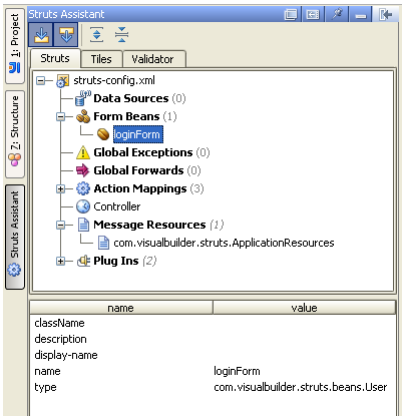
The structure of a Struts application including numerous Struts elements and the interaction between them is defined in the Struts configuration file `struts-config.xml`.
The Struts configuration file is created automatically when you enable Struts support for a module.

With IntelliJ IDEA, you can edit the `struts-config.xml` in the Struts Assistant tool window, which provides three synchronized views: the Structure Tree, the Properties Table, and the Struts Web Flow Diagram.

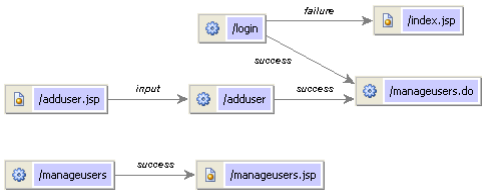
To define the structure of a Struts application, create and edit the necessary Struts elements and specify interaction among them using the Struts Assistant tool window, tab Struts. This dedicated tool window provides three synchronized views:

- the Structure Tree to add, edit, and remove Struts elements.

- the Properties Table to specify the values of elements and their attributes.



- the Struts Web Flow Diagram to navigate through the Struts application structure.



IntelliJ IDEA provides the facilities for managing the following Struts elements according to the common procedure:

- Data Sources
- Form Beans
- Global Exceptions
- Global Forwards
- Action Mappings
- controller
  There can be only one controller element in a Web application.
- Message Resources
- Plugins

Managing tiles and validators in a friendly interface requires that you enable support of them first.

Managing Struts Elements - General Steps

IntelliJ IDEA provides a user-friendly visual interface for managing Struts elements which includes:

- Creating , editing , and removing elements.
- Handling attributes of elements.
- Specifying and editing elements' properties and attributes .

The set of actions and attributes available for a specific element depends on its nature.

To create an element

All types of elements are created in the same way.

1. Open `struts-config.xml` .
2. Switch to the Struts Assistant tool window , tab Struts.
3. Right-click the corresponding element type in the Structure Tree and select the Add action from the context menu.
4. Specify the properties of the new element in the Properties Table.

Alternatively, right-click the element in the tree and select Jump to Source or press F4 . This will bring you to `struts-config.xml` in the text view where you can specify the element's properties manually. IntelliJ IDEA displays a template for specifying the properties that are mandatory for the elements of the specific type.

To remove an element

All types of elements are removed in the same way.

- Right-click it in the Structure Tree and select the Remove action from the context menu.

To view or edit an element

1. Select it in the Structure Tree or on the Struts Web Flow Diagram and make the necessary changes in the Properties Table.
2. To add an attribute to the element, right-click the element in the Structure Tree and select Add in the context menu.
3. Specify the properties of the attribute in the Properties Table.
4. To remove an attribute, right-click it in the Structure Tree and select Remove in the context menu

> Warning! When you remove an attribute, its nested properties are removed as well.

Managing Tiles

Tiles is a templating system to create a common look and feel for a Web application.

Tiles are managed through creating, editing, and removing tile definitions, which describe tile attributes and tile elements. Tile definitions are specified in the `tiles-defs.xml` which is created when you enable tiles support.

With IntelliJ IDEA, you can edit the `tiles-defs.xml` file in the Struts Assistant tool window , which provides two synchronized views: the Structure Tree and the Properties Table .

Tiles are removed in the same way as other Struts elements.

Editing a tile includes editing its properties and handling its attributes : forwards, exceptions, and set properties.

To create a tile

1. Enable tiles support .
2. Open `tiles-defs.xml` .
3. To add a tile definition, add the `<definition>` element to `tiles-defs.xml` manually.
4. Switch to the Struts Assistant tool window , tab Tiles. A new Definition node appears in the Structure Tree.
5. Specify the attributes of the new tile in the Properties Table.
   Alternatively, right-click the tile in the tree and select Jump to Source or press F4 . This will bring you to `tiles-defs.xml` where you can specify the tile's attributes manually. IntelliJ IDEA displays a template for specifying the mandatory attributes.

To remove a tile

– Right-click the tile in the Structure Tree and select Remove Definition from the context menu.

To edit a tile

1. Select it in the Structure Tree.
2. To edit the attributes of the tile, make the necessary changes in the Properties Table .
3. To manage the elements of a tile , use the Structure Tree and the Properties Table.

The elements are created, edited, and removed in the same way as common Struts elements.

Managing Validators

The Validator functionality within the Struts framework is intended for validating the data of forms in a Web application.

The Validator functionality uses two `xml` configuration files:

- The `validator-rules.xml` file defines the standard reusable validation routines that make the basis for configuring form-specific validations. The file is created when you enable Struts support and is supplied with a predefined set of commonly used validation rules such as Required, Minimum Length, Maximum length , Date Validation, Email Address validation and others.
  The file is closed for editing.
- In the `validation.xml` file, define the validations applied to a particular form bean.

While you edit `validation.xml` , IntelliJ IDEA coordinates the code of the two files and provides you with coding assistance based on the routines defined in `validator-rules.xml` .

To define the validations to apply to a form

1. Open `validation.xml` .
2. To add a validation definition, add the `<form-validation>` tag manually.
3. Switch to the Struts Assistant tool window , tab Validator. A new node appears in the Structure Tree.
4. Edit the attributes of the validation definition in the Properties Table.

Using the Web Flow Diagram

Use the Web Flow Diagram for navigating through Struts configuration file `struts-config.xml` and viewing Struts actions, pages, and other elements and interactions among them. The Diagram is coordinated with the Structure Tree and Properties Table views.

To see the Web Flow diagram

- Open your Struts configuration file `struts-config.xml` and click Diagram at the bottom of the editor window.

The layout of the diagram is common for all diagrams .

Struts 2

In this section:

- Struts 2
  - Introduction
  - Developing Web applications with Struts 2
- Preparing to Use Struts 2
- Managing Struts 2 Elements

Introduction

IntelliJ IDEA supports integration with the Struts 2 framework for creating Java EE and Web Applications.

IntelliJ IDEA provides the following support of the Struts 2 framework:

- Automatic creation of the specific module structure and the Web application deployment descriptor `web.xml` based on the dedicated Web facet.
- Downloading and adding the necessary libraries automatically and creating the Struts 2 application configuration file `struts.xml` based on the dedicated Struts 2 facet .
- A file set manager , library validator, and automatic detection of configuration settings.
- Coding assistance, including code completion, syntax and error highlighting, documentation lookup, and refactoring.
- Multiple code inspections that spot code issues specific for Struts 2.
- Smart navigation across Java, JSP and XML files.
- Support for inplace JavaScript and CSS code.
- Graphic view for the `struts.xml` configuration file.
- Java EE preview for JSP and JSF pages.
- Structure tool window that provides navigation across the native Struts 2 application structure and displays Struts 2 actions.

Developing Web applications with Struts 2

To develop a Web application using Struts 2, follow these general steps

1. Enable support of Web and Struts 2 development.
2. Create and configure general Web application elements : servlets, filters, and listeners.
3. Create and configure Struts 2 specific elements .
4. Populate your application with the necessary classes.
5. Deploy and run your application.

Preparing to Use Struts 2

On this page:

- Before You Start
- Basics
- Creating a Module with a Struts 2 Facet
- Adding a Struts 2 Facet to a Module
- Defining the Validation File Set
  - Adding a Validation File Set
  - Editing a Validation File Set

Before You Start

Make sure the Web and Struts2 bundled plugins are enabled. The plugins are bundled with IntelliJ IDEA and activated by default. If the plugins are not activated, enable them on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Basics

Integration with Struts 2 is enabled through the Web and Struts 2 facets. These dedicated facets contain settings, configuration file paths, and validation rules. This information determines the structure of a module so IntelliJ IDEA detects how to treat the module contents. See Facet for details.

A Struts 2 facet can be only added as a child of a Web facet . Note that only one Struts 2 facet is allowed in a module.

Creating a Module with a Struts 2 Facet

When you create a Struts 2 module with the dedicated Web and Struts 2 facets, IntelliJ IDEA configures the module, downloads and adds all the necessary libraries automatically.

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java . In the right-hand part of the page, specify the JDK that you are going to use.
3. Under Additional Libraries and Frameworks, select the Web Application check box.
   Select the version of the Servlet specification to be supported from the Versions list.

   If you want the deployment descriptor `web.xml` file to be created, select the Create web.xml check box.
4. Select the Struts 2 check box.
5. You'll need a library that implements Struts 2. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the library files that implement Struts 2. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

   Click Next.
6. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

IntelliJ IDEA configures the new module as follows:

- Creates a web node with a web application descriptor `web.xml` .
- Creates a Struts 2 configuration file `struts.xml` and adds it to the src node.
- Adds the required Struts 2 libraries.

Adding a Struts 2 Facet to a Module

When you add a Struts 2 facet to an existing module, IntelliJ IDEA downloads and adds the necessary libraries automatically.

A Struts 2 facet can be added only as a child of a Web facet. If the module has no Web facet, add it first. Also note that only one Struts 2 facet in a module is allowed.

This will change the configuration of the module .

1. Open the Module Settings dialog box .
2. In the Modules node, right-click the relevant module.
3. To add a Web facet, choose New from the context menu. In the drop-down list of available facets, select Web.
4. Right-click the Web Facet node and select New on its context menu.
5. Select Struts 2 on the context menu of available facets. The right-hand pane displays the Facet Struts 2 dialog.
6. In the File Sets tab, specify the configuration files to validate .
7. If some additional libraries are required, the corresponding warning message displays. Click Fix. The necessary libraries will be downloaded and added automatically.

The created Struts 2 configuration file `struts.xml` is located in the src node.

Defining the Validation File Set

After you have enabled the Struts 2 support, IntelliJ IDEA provides the ability to define the validation list by adding , editing , and removing file sets .

Adding a Validation File Set

1. Open the Struts 2 Facet page as for changing the facet settings .
2. Switch to the File Sets tab.
3. Click + (`Alt+Insert`). The Edit File Set dialog opens showing the module tree.
4. In the File Set Name field, specify the name of the new file set.
5. To add a file to the set, select the check box next to its name.
6. To add a file that is not displayed in the tree, click Locate and select the the required files in the Select Path dialog .

Editing a Validation File Set

1. Select the file set on the File Sets tab and click (`Enter`). The Edit File Set dialog opens.
2. To add files to the file set, select the check boxes next the required files and click OK.
3. To remove a file, select the file and click — (`Alt+Delete`).

To remove a validation file set, select it on the File Sets tab and click — (`Alt+Delete`).

Managing Struts 2 Elements

Struts 2 elements like actions, results, and interceptors as well as interaction between them is defined in the `<package>` section of the `struts.xml` configuration file. The file is created automatically when you enable Struts 2 support for a module.

IntelliJ IDEA provides you with a friendly interface for editing `struts.xml` both in the text view and in diagram .

To switch between the views

– Open `struts.xml` in the project view and click Graph in the bottom of the window.

To create a Struts 2 application element

1. Switch to the Project tool window and select Project in the View as drop-down list.
2. Right-click the struts.xml node and select New in the context menu. Then select the relevant element type. The New... dialog box opens.
3. Specify the element data and click OK. A new element is created and displayed in the project tree. The corresponding source code is generated in the `struts.xml` .

To remove a Struts application element, do one of the following

– In the Project tool window , select the element in the project tree. Then select Delete in the context menu.

– Switch to the `struts.xml` and make the necessary changes manually.

Swing. Designing GUI

In this section:

- Swing. Designing GUI
- GUI Designer Basics
  - GUI Designer Files
  - Bound Class
  - GUI Designer Output Options
- Customizing the Component Palette
  - Adding GUI Components and Forms to the Palette
  - Configuring Libraries of UI Components
  - Creating Groups
- Designing GUI. Major Steps
  - Binding the Form and Components to Code
  - Creating and Opening Forms
  - Localizing Forms
  - Making Forms Functional
  - Populating Your GUI Form
  - Setting Component Properties
  - Previewing Forms

GUI Designer Basics

IntelliJ IDEA's GUI Designer enables you to create graphical user interfaces (GUI) for your applications, using Swing library components. The tool helps you speed up the most frequently needed tasks: creating dialogs and groups of controls to be used in a top-level container such as a JFrame. When you design a form with the GUI Designer, you create a pane rather than a frame.

Dialogs and groups of controls created with the GUI Designer, can coexist in the application with the components that you create directly with Java code.

In this part:

- GUI Designer Files .
- Bound Class .
- GUI Designer Output Options .

Using the GUI Designer is subject to the following limitations:

- GUI Designer does not support modeling of non-Swing components.
- The GUI Designer does not create a main frame for an application, nor does it create menus.

GUI Designer Files

At design time, GUI information is stored in special files with a `.form` extension. These files are XML files that conform to a special schema. You can put the `.form` files into any version control system. A form can be associated with a Java source file .

The GUI Designer suggests two ways of producing GUI forms: you can either create them from scratch , or capture snapshots of the existing dialogs and create GUI Designer Forms from the snapshots. It doesn't matter if the dialogs were hand coded or made with another UI design tool - just capture them from the running application using the Form Snapshot feature.

A snapshot is not related to its original implementation. When you take a snapshot, you thus give up the source code used to produce the form.

Warning! IntelliJ IDEA does not provide the means to open the XML source of the `.form` files in the text editor, and you should not modify these files manually as text.

Bound Class

By default, IntelliJ IDEA automatically creates a Java class at the same time it creates a new GUI form. The new form automatically binds to the new class via the bind to class property. When components are added to the design form, a field for each component is automatically inserted into the source file of the Form's class (with some exceptions such as JPanels and JLabels which don't automatically get field names when placed on a Form). The field name appears in the relevant component's field name property, thereby binding the component to code in the class. It only remains to augment the generated code with whatever additional code is needed to implement behaviors and functionality of the form.

Tip Automatic binding of a component to a field is stipulated by the Create binding field automatically property of a component in the Palette.

You can opt to create a new form without a bound class. In this case you will have to explicitly bind a form to the source code .

GUI Designer Output Options

IntelliJ IDEA's GUI Designer provides two options for the output:

- Runtime classes that are generated when you compile the project to which they pertain. In this case, the intermediate step of Java source code is bypassed. Optionally, the compiled classes can be written to the compiler output directory configured for the project.
- Java source code. In that case, all code for the UI components is generated to the class to which each form is bound.

Refer to the GUI Designer dialog for the detailed description of options.

Customizing the Component Palette

You can customize the GUI Designer's component Palette, adding groups to organize your own custom UI components and/or components from one or more third-party libraries. You can also edit the contents of any existing Palette groups to add or remove components.

To customize your Palette, perform the following general steps

1. Install the desired archives of UI components in a location accessible to your computer. One of the possible locations is the `lib` folder of your IntelliJ IDEA installation.
2. Open the Project Settings dialog .
3. Configure the archives of UI components .
4. In the GUI Designer Palette, create groups and add UI classes or forms .

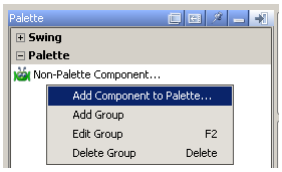
Adding GUI Components and Forms to the Palette

Adding new elements to the Palette enables you to reuse components from the libraries, or already created GUI forms that exist in your project.

For visual identification of the new components, you can use icons. Icons should not be larger than 18 x 18 pixels. If no icon is provided for a component, the Palette will default to the icon of the ancestor class if the component is derived from a Java UI class, or a default icon.

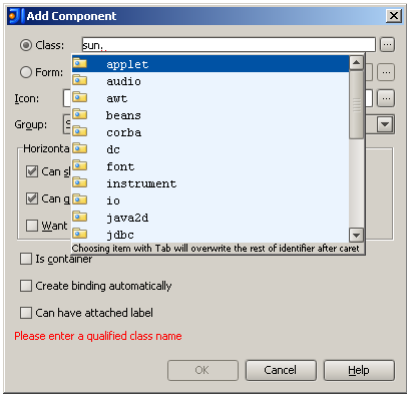
To add a new component to the Palette

1. Right-click the target group, and choose Add Component to Palette. The Add Component dialog opens.

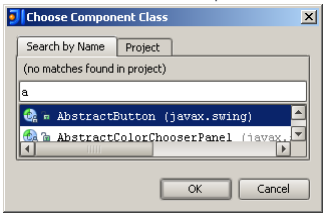


Tip To change an existing component, select one in the Palette, and choose Edit Component command on the context menu.

2. Click Class radio-button to add a component from a class library, or Form to add an existing GUI form.



3. Specify the fully qualified name of the component class or form, or click the ellipsis button next to the selected field, and choose the desired component class or form file in the Choose Component Class or Choose Form File dialog respectively.



Note If you are adding a form to the palette, make sure that the top-level component of the form is bound to a field: the field name property in the Inspector should not be void.

4. Optionally specify an icon for the component. Enter the fully qualified name of the icon file, or click the Browse button next to the Icon field, and and choose the desired icon in the Choose Icon File dialog.

5. In the Horizontal / Vertical size policies section, define how the component should behave when its parent container is being resized .

6. Set the options Is container , Create binding automatically , and Can have attached label .

7. Click OK to add the component to the target group.

Configuring Libraries of UI Components

You can use UI beans from the custom libraries accessible to the module in which you want to use the components. You have to decide whether to specify each library as a single-entry, global or project level. If you will use a UI library in multiple projects, add it to the global libraries. If you have UI classes that will be used in a single project only, add the library to the project libraries for the relevant project.

Refer to the section Configuring Module Dependencies and Libraries for the detailed description of procedure.

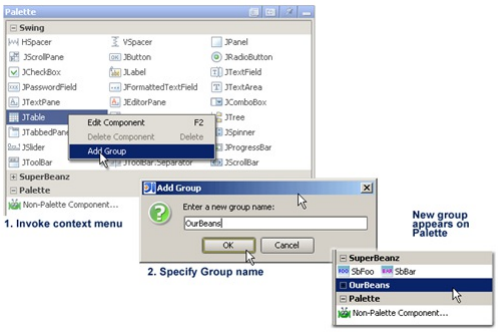
Creating Groups

The GUI Designer's Palette tool window can contain groups which can be used as a means of organizing GUI components. Each Group can contain one or more GUI components, and can optionally display a custom icon for each component.

Initially, there are 2 default groups: Swing that contains the default `javax.swing` components, and Palette that contains one default node: Non-Palette Component. You can create other Groups for components from your libraries of GUI components and forms.

To create a new Group

1. Right-click anywhere in the Palette tool window and choose Add Group on the context menu.
2. In the Add Group dialog, type a name for the new Group.
3. Click OK. A new group with the specified name adds to the Palette.



Designing GUI. Major Steps

The process of designing a graphical user interface involves the following typical steps:

- Creating a new form or dialog
- Placing and arranging components in the form
- Defining properties of the components
- Binding components to the source code
- Making forms functional
- Localizing forms
- Previewing Forms

Tip You might want to review basic concepts , and take a brief tour of the tool , if you have not already done so.

Binding the Form and Components to Code

If you have not created a bound class by default, when creating a new form, you do it explicitly, as described below.

There are two basic ways you can approach explicit binding depending on your situation:

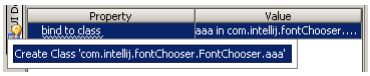
- Bind a form to an existing class , and its components to the fields of this class.
- Bind a form and its components to a class and fields that don't yet exist .

Binding a Form to a New Class

When all components are cleared, or the Form node is selected in the Component Treeview, the form's properties the Inspector displays properties of the form.

To bind a GUI form to a not-yet-existing class

1. Select the desired form in the Form Workspace or in the Components Treeview.
2. In the bind to class field, type the fully qualified name of the new class. Intention action icon appears suggesting you to create the specified class.
3. Click the intention action icon, or press `Alt+Enter`. The desired class is created in the specified location.



Binding a Form to an Existing Class

When all components are cleared, or the Form node is selected in the Component Treeview, the form's properties the Inspector displays properties of the form.

To bind a GUI form to an existing class

1. Select the desired form in the Form Workspace or in the Components Treeview.
2. Do one of the following:
   - In the bind to class field, type the fully qualified name of the bound class.
   - Click the Browse button and choose a class in the Choose Class to Bind dialog box.

Binding Groups of Components to Fields

When you create a button group in a form that is bound to source code, no field for the group is created. If you need to reference the group in code, you will have to explicitly bind the group to a field.

To bind a button group to a field

1. Select the button group in the Component Treeview. The group's properties appear in the Inspector pane.
2. Make sure the value in the Name property is the name you want for the new field in the source code.
3. Check the Bind to field check box to generate the field declaration in the source file bound to the parent GUI form.

Tip You can remove the field binding by clearing the Bind to field check box. This removes the field in the source code file.

Creating and Opening Forms

GUI Designer equips the developers with the possibility to create GUI forms with the desired layout and bound class, and dialogs. A form can be created as only a form file, or together with a UI class. A dialog framework consists of a class defining basic components and methods, and a GUI form with components bound to the fields of the UI class.

Tip As mentioned in the GUI Designer concepts , GUI forms in IntelliJ IDEA are not Java classes. They are special XML files stored in your project with a `.form` extension.

Prior to creating GUI forms and dialogs, select a package where the forms will be stored. You might need to create a package first. It is recommended to store the forms and dialogs in the same package as the classes that implement their functionality.

Creating a New GUI Form or Dialog

To create a new GUI form

1. On the main menu, choose File | New, as described in the section Creating Files .
2. On the pop-up menu, choose GUI Form.
3. Specify the name of the new form, and select the layout manager.
4. Check the option Create bound class, and specify the bound class name.
5. Click OK.

To create a new dialog

1. On the main menu, choose File | New, as described in the section Creating Files .
2. On the pop-up menu, choose Dialog.
3. Specify the name of the new dialog.
4. Specify the following options:
   - Generate `main()` method that creates and shows dialog.
   - Generate OK handler.
   - Generate Cancel handler.
5. Click OK.

Creating a form from a File Template

You can save any form as a File Template, and thus preserve commonly needed dialogs or panes of controls for later reuse. For detailed information about creating and using File Templates, refer to the sections:

- Creating and Editing File Templates
- Creating Template-based Files

To create a File Template from a GUI form

1. Create a GUI form or open it in the GUI Designer.
2. On the Main menu, choose Tools | Save File as Template. The File Templates dialog displays the source code of the form. The extension of the new template is `.form`
3. Specify the template name, and click OK.

To create a new GUI form from a File Template

1. In the Project tool window, right-click the package where you want to create the new form.
2. On the context menu choose New. The submenu contains items for all defined File Templates, including GUI forms.
3. Select the desired GUI form template on the context menu, and give it a name when prompted.

Creating Snapshots

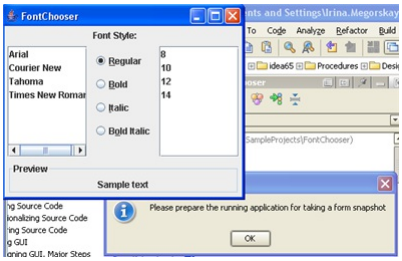
The Snapshot feature enables you to convert GUI forms created by some other means into IntelliJ IDEA GUI Designer Forms. All you need to do is run your application and save a part of the running application's component tree as a GUI Designer form. From then on, you can use the GUI Designer to extend or modify the form.

You can work with the layout and code in the GUI Designer just as you do with Forms originally created with the GUI Designer.

To take a Snapshot of an existing dialog

1. Open the module that contains source code for the dialog you want to capture.
2. Select the Application Run Configuration , make sure that the option Enable capturing form snapshots is checked, and run the application .
3. In the running application, perform whatever interaction is necessary to open the dialog you want to capture.
4. In the Project tool window, right-click the target package and choose New | Form Snapshot on the context menu. If the application is not yet running, IntelliJ

IDEA prompts you to run it, and then prepare the application for taking snapshot:



The Create Form Snapshot dialog box appears.

5. In the IntelliJ IDEA's Create Form Snapshot dialog, select the top-level component to be included in the snapshot, enter the form name, and click Create Snapshot button. The new form created this way is not bound to any class. You have to bind it to a class yourself.

Note If the selected node is not a valid container for a component, the Create Snapshot button is disabled.

Opening Existing GUI Forms

Open form files the same way you open source code or text files for editing. Locate the file in the appropriate navigation view, select it, and do one of the following:

- Double-click.
- Choose Jump to Source on the context menu.
- Press F4.

Each form opens in a separate tab in the same space occupied by the Editor for source code files. You can have source code/text files and form files open simultaneously.

Localizing Forms

The GUI Designer extends I18N Support to the GUI forms, and enables you to find components that contain hard-coded strings and transfer these strings to properties files. Having produced the various locales, you can change them at design time and preview how the localized forms will look like.

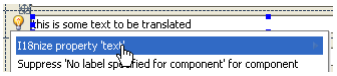
In this section, you can find general recommendations for applying i18n support to GUI forms:

- General localization procedure
- Suppressing I18nize Hard-Coded String quick fix
- Changing design-time locale

General Localization Procedure

To localize your GUI forms, perform the following major steps

1. Make sure that Hard-Coded Strings inspection is activated. Refer to the section Recognizing Hard-Coded String Literals for details. If the inspection is activated, the appropriate tool tips for components display in the UI Designer tool window, and quick fix i18nize property<name> is displayed in the Form Workspace and UI Designer tool window.
2. Create properties files for each locale you want to provide for the GUI, and a default properties file without a postfix.
   For example, for a dialog developed in English and localized to German and French, you might create the following properties files:
   1. `Subscription.properties`
   2. `Subscription_de.properties`
   3. `Subscription_fr.properties`
3. Go through the form, and select each component that contains hard-coded strings. Click the quick fix icon, or press `Alt+Enter` to show the list of available intention actions, and select i18nize property <name>, as shown on the following image:

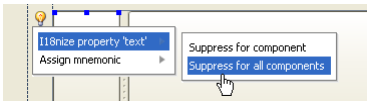


4. In the I18nize Hard Coded String Literal dialog box, extract hard-coded string literals to the specified properties files. Refer to the section Extracting Hard-Coded String Literals for details.
5. Edit the extracted strings in the resource bundle. Refer to the section Editing Resource Bundle for details.
   See samples in the section Using Resource Bundles for Components .

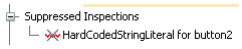
Suppressing I18nize Hard-Coded String Quick Fix

To suppress i18nize quick fix

1. Select a component that contains hard-coded text property.
2. Click the quick fix icon, or press `Alt+Enter` to show the list of available intention actions.
3. Hover the mouse cursor over the i18nize property <name> quick fix, and click the right arrow icon ▸ (or press right arrow key) to reveal the submenu, as shown on the following image:



4. Click the option Suppress for component or Suppress for all components. The list suppressed inspections appears in the GUI Designer tool window:



Tip To cancel suppression, select the desired inspection under the Suppressed Inspections node, and press `Delete` key.

Changing Design-Time Locale

To change locale at design time

1. With the GUI Designer tab having the focus, click the Change Local combo box <default> on the main toolbar.
2. Select the desired locale from the drop-down list.

Making Forms Functional

Note It is assumed that you are familiar with UI programming in Java. This section covers just the basics you need to add functionality to the forms you build with the GUI Designer.

In this section:

- Creating and Disposing of a Form's Runtime Frame
- Creating Form Initialization Code
- Creating Listeners
- Generating Accessor Methods for Fields Bound to Data

Creating and Disposing of a Form's Runtime Frame

To make your form work, you have to provide a runtime frame for it. The main() method for GUI forms takes care of creation and disposal of such frame.

To create a main() method for a form

1. Open for editing the bound class of a form where you want to create the main() method.
2. Press `Alt+Insert`.
3. On the pop-up menu, click Form main(). The following method is added to the source code:

```
public static void
                    main(String[] args)
                    {
                    JFrame frame = new JFrame("<class name>");
                    frame.setContentPane(new
                    <class name>().contentPane);
                    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    frame.pack();
                    frame.setVisible(true);
                    }
```

Creating Form Initialization Code

When you finish building a GUI Designer form , at a minimum you should have a class that is bound to the form, and component type fields in the class bound to the various form components.

If the binary class files are specified as the output option, all runtime initialization code is generated in the class file. If your form has a bound class, you will not see the automatically generated initialization code there. In case you have selected Java source code as the output option, the bound class of your form will contain automatically generated `$$$setupUI$$$()` method.

Warning! Do not edit manually the GUI Initializer source code! At any compilation, IntelliJ IDEA automatically generates new code in this section, and all your changes will be lost.

Sometimes you might need to provide initialization code of your own. For example, you want a GUI component to be instantiated by a non-default constructor with certain parameters. In this case, IntelliJ IDEA will not generate instantiation of the component, and it is your responsibility to provide the call to constructor in the `createUIComponents()` method. Otherwise, a Null Pointer Exception will be reported. Follow the general technique described below.

To create custom GUI initializer source code for a certain component, follow this general procedure

1. Select the desired component.
2. In the Inspector, check the option Custom Create.
3. With the component selected, press `F4`, or choose Jump to Source on the context menu.
4. In the text editor, locate the method `createUIComponents()`, and type the desired source code. The code in this method will not be removed on compilation.

Example

For example, you would like to provide non-default constructors for the radio buttons 1 and 2, and have GUI Designer create a default constructor for the radio button 3:

```
...
                    //For the radio buttons 1 and 2, option Custom Create is set to true.
                    //You write custom constructors for these components
                    //in the method createUIComponents()
                    private JRadioButton radioButton1;
                    private JRadioButton radioButton2;
                    //For the radio button 3 the default constructor is generated automatically
                    //in the method $$$setupUI$$$(). The component properties
                    //specified in the GUI Designer
                    //are generated as calls to the set* methods in $$$setupUI$$$().
                    private JRadioButton radioButton3;
                    ...
                    private void createUIComponents()
                    {
                    radioButton1 = new JRadioButton("Custom text 1");
                    radioButton2 = new JRadioButton("Custom text 2");
                    }
                    ...
                    private void $$$setupUI$$$()
                    {
                    createUIComponents();
                    ...
                    radioButton3 = new JRadioButton();
                    radioButton3.setText("RadioButton");
                    ...
                    }
```

Creating Listeners

GUI Designer enables you to create listeners for components. The components that have listeners, are marked with a special icon that shows up when such component is selected. From the components in the Form Workspace and in the Components tree view, you can quickly navigate to the respective source code.

To create a listener

1. Select component in the Form Workspace, or in the Components tree view.
2. On the context menu, choose Create Listener, or press `Ctrl+O`.
3. From the Create Listener pop-up menu, select the desired listener type.
4. In the Select Method to Override dialog box, select the desired method. Optionally, specify whether you want the JavaDoc comment for this method to be copied to the target file, and press OK. The stub method that overrides the specified method, is added to the bound class.
5. If you would like modify the body of the stub method, edit the Overridden Method Body file template:
   In the File Templates dialog , open the Code tab, and edit the template as required.

To navigate from a component to listener

1. Select component in the Form Workspace, or in the Components tree view.
2. On the context menu, choose Go to Listener.
3. In the Navigate to Listener pop-up menu, click the desired listener, or press `Ctrl+Alt+B`.

Tip In the Form Workspace, click and select the desired listener.

Generating Accessor Methods for Fields Bound to Data

If you have fields in the UI class that you want bound to some data in a JavaBean class, IntelliJ IDEA provides a special Data Binding Wizard . This wizard helps generate getData and setData methods for the fields in a UI class that are bound to components in a GUI form. You can create data binding to an existing bean class, or create a new bean class on the fly using the wizard.

For example, suppose you have some JTextField components on a GUI form that are bound to fields in a UI class, and you want the value of their text properties bound to some data in a JavaBean class. Instead of editing the UI class, creating getData and setData methods for the fields, being careful to specify the fully qualified name of the correct bean class and importing the required package, just use the Data Binding Wizard.

Note The GUI form must be bound to a class, and the GUI components must be bound to some fields in the class before running the Data Binding Wizard.

When you finish the wizard, the specified accessor methods are written to the form's class according to the options you specified in the wizard. Import statements are generated in the class as necessary.

Once components have been bound to data, you can invoke the Data Binding Wizard again, in which case you are presented with options to either remove the data binding, or bind to a different bean. If you choose the latter option, the wizard enables you to select or create the bean class for the data binding. The source code of the UI class (including imports) is modified according to your choices and specifications.

Tip If you look at the code and discover you made a mistake, you can use Undo to revert the code to its former state.

To bind a component to data

1. Make sure that your form contains components that are eligible for data binding.
2. On the context menu of the form, choose Data Binding Wizard, or click on the GUI Designer toolbar.
3. On the first page of the wizard, specify whether to bind to data from a new bean or an existing bean, specify the class for the option you choose, and click Next.
4. The second page lists the form components that are eligible for data binding, and that have been bound to a field in the form's UI class. For each component name, click the Bean Property field to the right, and specify the bean property that you want as the data source/target for the component.
5. Click Finish to complete the data binding operation.

Populating Your GUI Form

This part explains how to place and lay out components on forms in the GUI Designer.

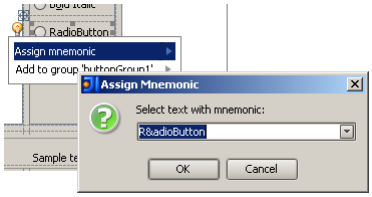
When you create a new GUI form, a top-level JPanel is automatically placed in the Form Workspace ready to receive the additional components that will make up the GUI you want to build. The pane's Layout Manager property is set to the default layout manager configured in the GUI Designer Options .

- The valid target areas where you can place a component, depend on the selected layout.
- Undo and Redo work for add, copy, move and delete operations in the GUI Designer.
- Use `Escape` to cancel any placement operation.

Adding Mnemonics

You can specify a mnemonic character for action controls by placing an ampersand (&) character in front of the desired mnemonic character in the text property. At runtime, the mnemonic character is underlined.

For components that can take mnemonics, there is an Intention Action Assign mnemonic that intelligently keeps track of already used mnemonics in the form and suggests a mnemonic for the current component based on what characters are already used.



To delete one or more components

1. Select one or more components in the Form Workspace or in the Components Treeview .
2. Press `Delete`.

Duplicating Components

IntelliJ IDEA's GUI Designer provides several ways to create new instances of the same component on a form without returning to the Component Palette.

To add multiple instances of a component from the Palette to a form

1. Click the desired component in the Palette .
2. Keeping the `Ctrl` button pressed, click the target locations in the form.
3. Release the `Ctrl` button when ready.

To create a new instance of a component in a new location

1. Select one or more components in the Form Workspace or in the Components Treeview .
2. Keeping the `Ctrl` button depressed, drag the selected component to the new locations.
3. Click each target location to create a new instance of the component.
4. Release the `Ctrl` button when ready.

To clone a component

1. Select the desired component on a form.
2. Press `Ctrl+D`.

Grouping and Ungrouping Components

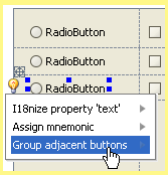
IntelliJ IDEA smartly detects components that are candidates for grouping together (for example, radio buttons). The first time you create a group, a button Groups node is added to the Component Treeview. A child node is added for each button group you add to the form.

Grouped buttons are visually indicated on the design form, when one of the buttons is selected in the Form Workspace or in the Components tree view, or when the whole button group is selected in the Button Groups node of the Components treeview:

To create a group of components

1. In the Form Workspace, select the components to be joined into a group.
2. On the context menu of the selection, choose Group buttons.
3. In the Group Buttons dialog box, specify the name of the new button group, and click OK .

Tip For the radio buttons, IntelliJ IDEA suggests a quick fix, which creates a group of adjacent buttons:



To remove grouping

Do one of the following:

– On the design form, select any component in a group, right click, and choose Ungroup Buttons.

– In the Component Treeview, right-click the relevant button group under Button Groups node, and choose Ungroup.

Morphing Components

Suppose you have created a JTextField component that contains some text, colors, font properties etc., and find out that you need a JFormattedTextField component with the same properties. This is when morphing comes to help.

Morphing enables you to transform selected component into a new one, while preserving all common properties.

To create a new component on the base of an existing one, preserving the common properties

1. Select the desired component in the Form Workspace or in the Components Treeview .
2. On the context menu, choose Morph Component.
3. In the Choose Component to Morph Into pop-up menu, click the target component type.

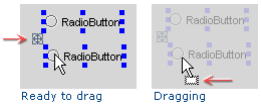
Moving Components

Components can be repositioned on a form however you wish. The results of any move depend on the setting of the Layout Manager property of the target container.

This section describes how to move components using drag-and-drop operation , or using navigation keys on your keyboard.

To move one or more components

1. Select one or more components in a Form Workspace .
2. A *drag handle* appears near the selected components indicating that you can drag to a new position on the form. (It is not necessary to point to the handle itself... hover anywhere over the selected components.)
3. Drag and drop the selection to the target container, or to the target area within the same container. The valid drop locations are highlighted with blue borders.



Tip You can also move components using the Components Treeview .

To move a component using keyboard

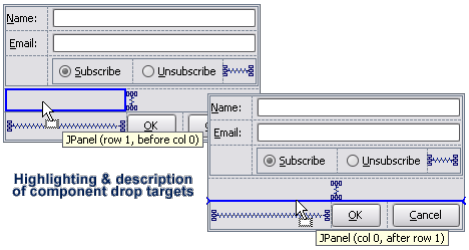
1. Select the desire component on the form.
2. Press `Ctrl+Arrow` .

Placing GUI Components on a Form

Use the Components Palette to populate your forms with the Swing GUI components.

To place a component from the Palette on a form

1. Click the desired component in the Palette tool window .
2. Move your pointer over the form workspace. Valid drop location in a container highlights when the component placement pointer hovers above it, the pointer changes its shape, and the drop target is described briefly in a tooltip, as shown in the following figure:



3. Click on the form workspace at the desired valid location to place the component.
4. Check the option Is container, if you want the new component to be able to accommodate nested components.
5. Check the option Create binding automatically, if you want to generate a field for the component in the bound class.
6. Click OK.

Placing Non-Palette Components or Forms

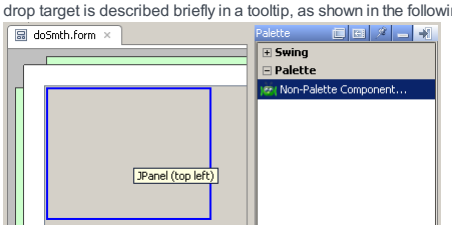
In addition to the GUI components from the Component Palette, you can use GUI components and forms not registered on the Palette, but available via your project's paths configuration.

A non-palette component can be associated with a GUI form, or with a class derived from JComponent. Such class should be available in your project, or in the libraries. The class should be compiled, and should have a default constructor, because GUI Designer instantiates objects by calling their default constructors.

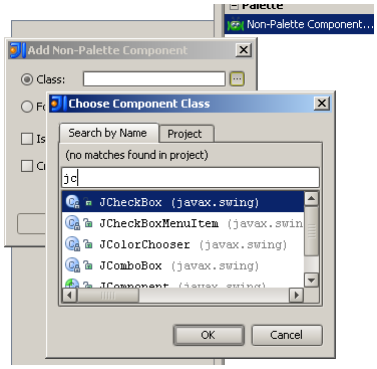
If a class does not have a default constructor, but instead has a constructor with parameters, the Custom Create property of this component is set to `true`, and `createUIComponents()` method is added to the source code, where you have to provide the form initialization.

To place non-Palette components on a GUI form

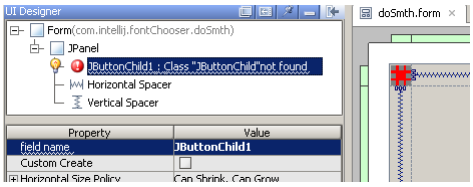
1. In the default Palette group, click the Non-Palette Component node.
2. Move your pointer over the form workspace. Valid drop location highlights when the placement pointer hovers above it, the pointer changes its shape, and the drop target is described briefly in a tooltip, as shown in the following figure:



3. Click on the form workspace at the desired location to place the component.
4. In the Add Non-Palette Component dialog box, click the Class or Form radio button, and specify the name of the class or form in the text field. Alternatively, click the corresponding ellipsis button, or press `Shift+Enter`.
5. If you have selected Class option, locate the desired class, using Search by Name tab, or Project tab:



If IntelliJ IDEA cannot locate the compiled class with the specified name and the default constructor, the component is red highlighted in the form, which means that it will not show at runtime:



Compile the class `Ctrl+Shift+F9` and press Reload Custom Components button on the toolbar

6. If you have selected Form option, locate the desired form, using Search by Name tab, or Project tab. The form should be bound to a class.
7. Check the option Is container, if you want the new component to be able to accommodate nested components. In this case, the component will acquire certain properties that are specific for containers.
8. Check the option Create binding automatically, if you want to generate a field for the component in the bound class.
9. Click OK.

To select one or more components

To select a single component, just click it. To select multiple components, do one of the following:

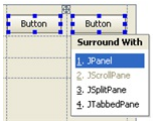
- In the Components Treeview , or in the Form Workspace , use `Shift+Click` to select multiple contiguous components, or `Ctrl+Click` for multiple non-contiguous components.
- Use arrow keys to move selection to the adjacent component.
- Use `Shift+Arrow keys` to select multiple contiguous components.
- Use `N/A` and `N/A` to expand or shrink selection within a container.

Wrapping/Unwrapping Components

The Surround With feature is available in the GUI Designer, as well is in the editor. In the GUI Designer, you can wrap a container of your choice around one or more components, and remove wrapping.

To wrap a container around components

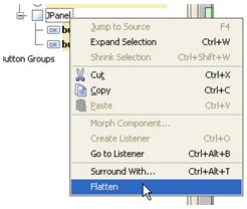
1. Select one or more components in the Form Workspace, or in the Components tree view.
2. On the context menu of the selection, choose Surround with, or press `Ctrl+Alt+T`.
3. In the Surround With popup menu, click the desired container, or press the corresponding numeric key.



When reasonable, IntelliJ IDEA smartly suggests you to wrap a component into a container. This is done with the help of intention action. For example, IntelliJ IDEA suggests to wrap a JList into a JScrollPane:

To unwrap components from the surrounding container

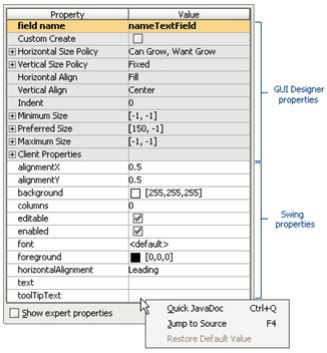
1. Right-click container node in the Components tree view.
2. On the context menu, choose Flatten.



Setting Component Properties

Whenever you select a component or an entire form in the Form Workspace, or in the Components tree view, the respective properties display in the Inspector. For the properties of different types (text, Boolean, color, etc.) the Inspector provides different editors. Besides editing in the Inspector , you can edit text properties using the in-place editor.

Find the detailed description of properties in the section Component Properties



To select a property in the Inspector, just click the respective property name. To activate the editor for the selected property, click the Value column. If the property requires choosing the value, the ellipsis button is revealed. Click this button to invoke one of the available property editors, or click `Shift+Enter` .

Use the following editors to modify properties in the Inspector:

- Text field: Type a value.
- Pick list: Pick a value from a drop-down list of valid choices.
- Check box: Set value for Boolean type properties.
- Dialog: Presents the Browse button ... that opens the relevant dialog box: Edit Text, Select Color, Choose Icon File etc. For text properties, refer to editing text property procedure .

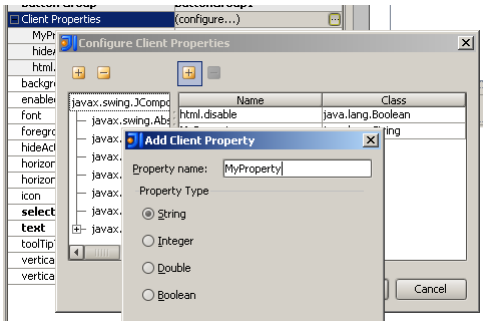
Configuring Client Properties

You can configure custom properties for the components of a GUI form. These are the properties listed under the Client Properties node of the Inspector . This section describes how to manage custom properties of the GUI components.

To configure a Client Property

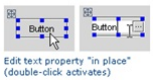
1. Select a component in a GUI form.
2. In the Inspector , click the Value column of Client Properties node to reveal the ellipsis button.
3. Click the ellipsis button in the edit field. The Configure Client Properties dialog is opened.
4. In the left-hand pane, select the class for which you want to change a Client Properties. If the class is not present in the list, click the + button above the list of classes and specify the name of the desired class.
5. With the class selected in the left pane, click the + button on the right pane. The Add Client Property dialog appears.

   Tip To delete a client property of a component, click — button.
6. Enter a name for the property in the Property name field using Java naming conventions.
7. Select the option that corresponds to the new property's type and click OK.



Setting Text Properties

After placing the component on the form workspace, the in-place editor for the Text property activates automatically.



To activate the in-place editor later, select the desired component and double-click it, or press `F2`. Finally, you can select a component and just start typing - in this case, the existing text will be lost.

If you need to enter a lengthy text, or take a value from a resource bundle, refer to the following procedure.

To edit a text property

1. Click the ellipsis button. In the Edit Text dialog box, specify whether you want to edit the text as a string or resource bundle.
2. If you have selected the String option, type the desired text in the Value field, and proceed to the step 4.
3. If you have selected the Resource bundle option, specify the name of the desired resource bundle , key and value.
   - Click ellipsis button next to the Bundle name field, or press `Shift+Enter`. In the Choose Properties File dialog box, find the desired resource bundle in the project tree view, or click the Search by Name tab, type the search string and select resource bundle from the suggestion list.
   - Click ellipsis button next to the Key field, or press `Shift+Enter`. In the Choose Property Key dialog box, select the desired key from the list.

   Type value in the Value field.
4. If you want to enable i18n support for the property, make sure that the option Value does not need internationalization is cleared. In this case, intention action i18nize property text is suggested for this component. If the option is selected, the intention action is suppressed.
5. Click OK.

Tip When you invoke the in-place editor for a JList or a JComboBox, an editor dialog box is opened, enabling you to type the elements of the list.

Previewing Forms

The GUI Designer provides a Preview function that enables you to see how the form will look at runtime. When the form runs in Preview mode, you can click buttons, check boxes, enter text, etc. just as at run time. Of course, there is no functionality at this point.

The Form Preview window enables you to change look and feel of the form, resize it, and exit the preview mode.

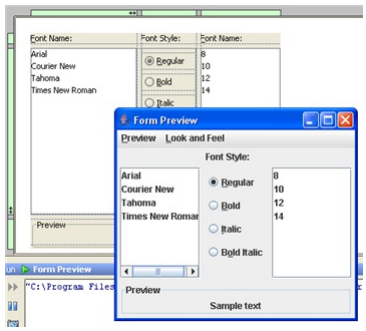
Note Form Preview shows only the properties specified by the GUI Designer in the `$$$setupUI$$$` method.

To preview a GUI form, do one of the following

- Right-click a form, and choose Preview on the context menu.
- With the form having the focus, click preview button on the main toolbar.

To exit the preview mode, do one of the following

- On the main menu of the Preview Form window, choose Preview | Exit.
- Click on the toolbar of the Preview Form window.



Tapestry

In this section:

- Tapestry
  - Introduction
  - Using Tapestry in IntelliJ IDEA
- Enabling Tapestry Support
- Creating Tapestry Pages, Componenets, and Mixins
- Tapestry Tool Window
- Editing Templates
- Tapestry View

Introduction

IntelliJ IDEA comes bundled with the Tapestry plugin that provides native support for developing Web applications based on the Tapestry framework. The plugin is by default enabled. If not, enable it on the Plugins page of the Settings dialog box.

IntelliJ IDEA allows you to quickly setup a running Tapestry project and provides several types of assistance in developing Web applications.

Using Tapestry in IntelliJ IDEA

To use Tapestry in developing a Web application, perform these general steps

1. Enable Tapestry support .
2. Create Tapestry-specific items : pages, components, and mixins.
3. Examine the structure of your application in the Tapestry-specific terms.
4. Edit the generated Tapestry templates .

Enabling Tapestry Support

You can enable Tapestry support when creating a project or module , or for an existing module.

- Enabling Tapestry support when creating a project or module
- Enabling Tapestry support for an existing module

Enabling Tapestry support when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Java Enterprise . In the right-hand part of the page, specify the JDK to be used and select the Java EE version to be supported.

3. Under Additional Libraries and Frameworks, select the Tapestry check box.

4. You'll need a library that implements Tapestry. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the library files that implement Tapestry. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

   Click Next.

5. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

6. Specify the application filter name and root package in the dialog that opens.

As a result, the Tapestry implementation library is added to module dependencies and a Tapestry facet is created.

Enabling Tapestry support for an existing module

1. Open the Project tool window (e.g. View | Tool Windows | Project).

2. Right-click the module of interest and select Add Framework Support.

3. In the left-hand pane of the Add Frameworks Support dialog that opens, select the Tapestry check box.

4. You'll need a library that implements Tapestry. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the library files that implement Tapestry. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

5. Click OK in the Add Frameworks Support dialog.

6. Specify the application filter name and root package in the dialog that opens.

Creating Tapestry Pages, Componenets, and Mixins

Tapestry items can be created only within the `components`, `pages`, or `mixins` dedicated packages. These packages, in their turn, should be located under the application root package.

In this topic:

– Enabling creation of Tapestry items
– Creating a Tapestry page
– Creating a component
– Creating a mixin

To enable creation of Tapestry items in a module

1. Open the Project tool window.
2. Switch to the Project or Package view by doing one of the following:
   – Select the necessary view from the list in the left-hand part of the title bar.
   – If the views are represented by tabs, click the corresponding tab.
3. Select the application root package, and choose New | Package on the context menu.
4. In the New Package dialog box, that opens, specify the name of the dedicated package depending on the type of items to be created under it.
   – pages for Tapestry pages .
   – components for Tapestry components .
   – mixins for Tapestry component mixins .

To create a Tapestry page

A Tapestry page normally consists of an HTML template that implements the page appearance and a Java class that implements the page functionality and behaviour. The HTML file has the same name as the Java file and the `.tml` extension. IntelliJ IDEA generates two stubs, with the `.tml` file containing the `<header></header>` and `<body></body>` sections.

1. Open the Project tool window.
2. Switch to the Tapestry view by choosing the Tapestry item from the list in the left-hand part of the title bar or clicking the Tapestry tab.
3. Right-click the pages node and choose New | Tapestry | Page on the context menu.
4. In the New Tapestry Page dialog box, that opens, specify the page name and the parent folders to store the generated class and template sources in.
   IntelliJ IDEA remembers the last selected directories and will automatically show them in the corresponding drop-down lists next time you create a page.
5. Customize the page generation, if necessary:
   – If you already have a page with the specified name and you want IntelliJ IDEA to overwrite it, select the Replace existing files check box.
   – To have only a Java class stub generated, select the Do not create template check box.

To create a new component

A Tapestry component normally consists of an HTML template that implements the appearance of a piece of a page and a Java class that implements the behavior and functionality of this item. The HTML file has the same name as the Java file and the `.tml` extension. IntelliJ IDEA generates two stubs, with the `.tml` file containing a `<div></div>` section.

1. Open the Project tool window.
2. Switch to the Tapestry view by choosing the Tapestry item from the list in the left-hand part of the title bar or clicking the Tapestry tab.
3. Right-click the components node and choose New | Tapestry | Component on the context menu.
4. In the New Tapestry Component dialog box, that opens, specify the component name and the parent folders to store the generated class and template sources in.
   IntelliJ IDEA remembers the last selected directories and will automatically show them in the corresponding drop-down lists next time you create a component.
5. Customize the component generation, if necessary:
   – If you already have a component with the specified name and you want IntelliJ IDEA to overwrite it, select the Replace existing files check box.
   – To have only a Java class stub generated, select the Do not create template check box.

To create a new mixin

1. Open the Project tool window.
2. Switch to the Tapestry view by choosing the Tapestry item from the list in the left-hand part of the title bar or clicking the Tapestry tab.
3. Right-click the mixins node and choose New | Tapestry | Mixin on the context menu.
4. In the New Tapestry Mixin dialog box, that opens, specify the mixin name and the parent folders to store the generated class in.
   IntelliJ IDEA remembers the last selected directory and will automatically show them it in the drop-down list next time you create a mixin.
5. Customize the mixin generation, if necessary:
   – If you already have a mixin with the specified name and you want IntelliJ IDEA to overwrite it, select the Replace existing files check box.

Tapestry Tool Window

This tool window shows contextualized information about some Tapestry element.

Documentation

This tab shows the documentation of a Tapestry component, page or mixin.

The documentation (where applicable) is composed of:

- General description
- List of required parameters (with their name, prefix, default value and description)
- List of optional parameters (with their name, prefix, default value and description)
- List of artifacts
- Examples
- Notes

When the element selected is not a Tapestry component, page, or mixin, the documentation is composed of:

- List of Tapestry Applications
- List of Core Library components, pages and mixins

Editing Templates

Loomy adds some additional features to the default HTML editor to allow faster and safer creation of Tapestry templates.

This section covers the following features:

- Syntax Highlighting
- Error Detection
- Auto-Completion
- Navigation

Syntax Highlighting

Special syntax highlighting in Tapestry templates in order to make template visual interpretation much easier.

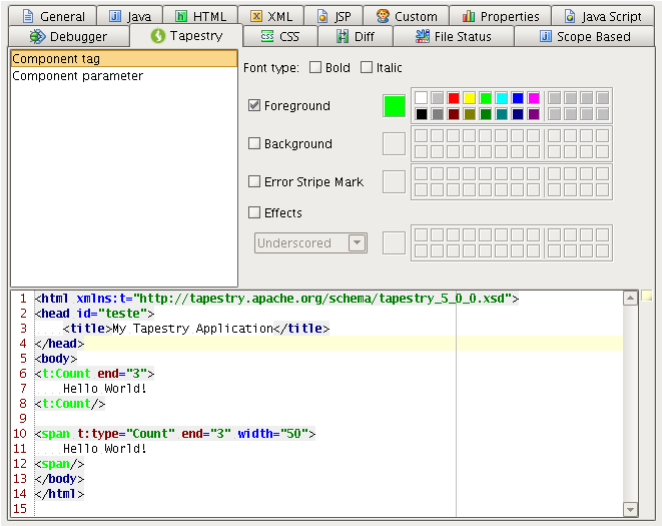
The section covers the following features:

– Configuration

– Component Tags Highlighting

Configuration

To configure the syntax highlighting feature you have to go to Colors & Fonts settings. In the Tapestry tab you can then choose the syntax coloring scheme that you prefer.



Component Tags Highlighting

You can configure the plugin to highlight every Tapestry specific tag and tag parameter in your template. If you do, you will be able to easily identify in your template what are normal tags and Tapestry tags.

```
<body>
<h1>Start Page</h1>

<p> This is the start page for this application, a good place to start your modifications.
    Just to prove this is live: </p>

<p> The current time is: ${currentTime}. </p>


<p>
    [<a t:type="PageLink" page="Home" name="homeLink">home</a>]
    [<t:pagelink page="Home" name="homeLink">refresh</t:pagelink>]
</p>
</body>
</html>
```

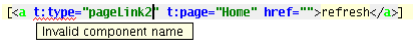
Tip If you get an inspection reporting invalid tag attributes then it's because you have an inspection that checks if your using any HTML tag parameter that your not supposed to. When using Tapestry you will have to use special tag parameters so if you don't want to get this warning just turn off the "HTML tags and attributes conventions" inspection that you can find under "HTML Inspection".

Error Detection

There are several errors that are only found when the application is running. With Loomy it's possible to detect many of these errors without losing the time to run the application.

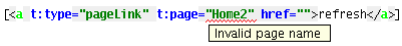
The plugin can detect the following errors:

Invalid component references

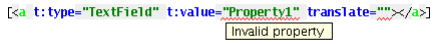


Missing required parameters

Invalid page names



Invalid property



Auto-Completion

Auto-completion makes template editing much faster. Loomy extends this feature introducing lot's of Tapestry specific completions.

This section covers the following features:

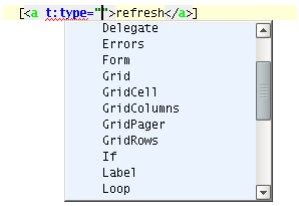
- Component Tags Auto-Completion
- Component Parameters Auto-Completion
- Parameters Values Auto-completion

Component Tags Auto-Completion

When editing your template you can take advantage of the auto-completion feature to help you insert a component using both types of possible annotations for that:

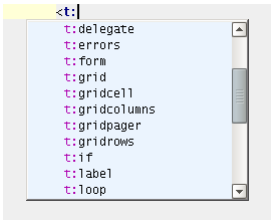
- Invisible Instrumentation

  Just use the auto-completion feature on the `t:type` attribute value.



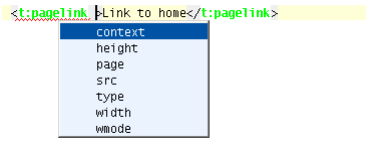
- Component Tag

  Just use the auto-completion feature on the tag name where you want to insert the component.



Component Parameters Auto-Completion

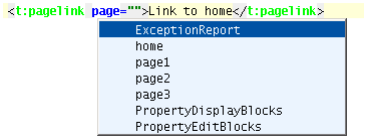
Component tags have additional parameters that don't conform to the HTML standard and so the default auto-completion won't help you. This plugin extends the auto-completion feature adding the Tapestry component parameters to the list of possible tag parameters.



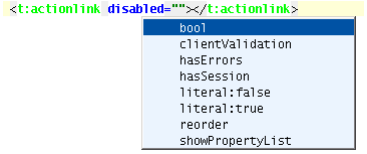
Parameters Values Auto-completion

Auto-completion of property binding is available so you can save lot's of time navigating from template to class.

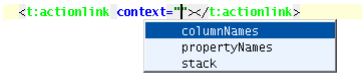
- Page parameter of the PageLink component



- Boolean parameters



- Simple properties



- Composed properties

Navigation

Loomy allows easy navigation between several elements in your template are the related content.

You can navigate:

- Between a template and the corresponding class.
- From a tag to the corresponding component class.
- From a property value to the associated method.
- To the component documentation.

Between template and class

Probably the most used navigation. Loomy provides an action (`Ctrl+Shift+G` by default) that allows you to navigate from a template to it's corresponding class and vice-versa.

From tag to component class

This navigation allows you to navigate from a component tag in your template to that component class. Just press `Ctrl+Click` on the tag itself.

From property value to associated method

If you have bound a component parameter with a property you can now just press `Ctrl+Click` on the parameter value in your template and you will navigate to the method that corresponds to the given property.

To component documentation

This navigation allows you to navigate from a component tag in your template to that component documentation displayed in the Tapestry ToolWindow. This action is associated by default to the key `Shift+Ctrl+D`, so just use when the cursor is on a component tag and you'll see it's documentation.

Tapestry View

In this view , you can browse your Tapestry application not in terms of classes and templates but in terms of pages, components, services, etc.

This section covers the following features:

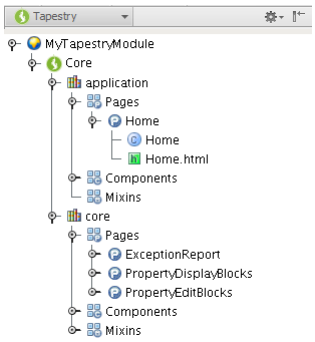
– Drag-and-Drop

– Safe Delete

General Description

This view is available to every project that has at least one web module with Tapestry support. It can be accessed like any other project view from the View as selection box.

The view let's you browse your Tapestry modules in a tree structure that shows you all Tapestry elements like pages and components.

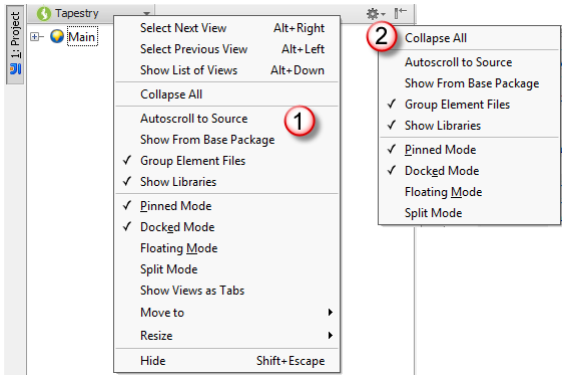


Structured View of the Application

Each element in the Tapestry View has a different icon so that it's easy to identify the type of element:

Library	**A Tapestry library.** **By default you will have two library nodes, one for your application and the other of the provided core library.**
Page	A Tapestry page. You can navigate to the page class or template from here.
Component	A Tapestry component. You can navigate to the component class or template from here.
Mixin	A Tapestry mixin. You can navigate to the mixin class from here.

The title bar context menu (1) provides additional functionality that will also help each user customize it's look&feel. Some options are also available from the list that opens when you click the button (2).



Below is a description of menu items that are specific for Tapestry.

Item	Description
Show From Base Package	Only Show Content From the Application Base Package
Group Element Files	Group Element Files Like it's Class and Template in a Parent Node.
Show Libraries	Show/Hide Tapestry Libraries.

Drag-and-drop

In order to increase development speed and prevent errors the plugin allows you to use drag&drop operations to generate simple code but that would otherwise be more error prone and take time from the task at hand.

This section covers the following features:

- Drag&Drop Components
- Drag&Drop Pages
- Drag&Drop Mixins

Drag&Drop Components

Drop in Page or Component Template

This drop operation will result in a reference to the dropped component being inserted at the drop point with all required attributes set.

Drop in Page or Component Class

This drop operation will result in the declaration of an embedded component in the target class.

```
@Component
private ActionLink _actionLink;
```

Drag&Drop Pages

Drop in Page or Component Template

This drop operation will result in a page link to the dropped page being inserted at the drop point.

```
<t:pagelink page="Home">Link to Home</t:pagelink>
```

Drop in Page, Component or Mixin Class

This drop operation will result in the injection of the dropped page in the target class.

```
@InjectPage
private Home _home;
```

Drag&Drop Mixins

Drop in Component Class

This drop operation will result in the declaration of the usage of an implementation mixin.

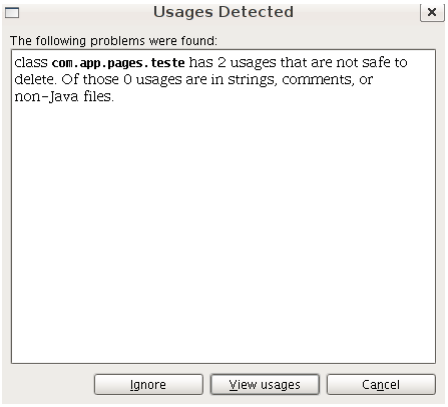
```
@Mixin
private MyMixin _myMixin;
```

Tip You will only be able to use the drag&drop feature if a valid drop target is in the currently open editor.

Safe Delete

If you execute the delete action from the Tapestry View you'll see that Loomy will actually figure out what Tapestry elements you're trying to delete and it will aggregate all it's file into this operation. So if for example you try to delete a page, Loomy will delete the page class, template and it's resource bundle.

IntelliJ IDEA will then look for usages of all the resulting files and you'll be able to, as usual, go through with the operation or cancel it.



Template Languages: Velocity and FreeMarker

On this page:

– Basics
– Associating template data languages with files and folders
– Creating an extension pattern for a template data language
– Using code completion
– Fixing unresolved references

Basics

IntelliJ IDEA lets you develop templates in Velocity and FreeMarker template languages (VTL and FTL).

Coding assistance for VTL and FTL is available for files whose file name extensions match the extension patterns associated with the Velocity Template and FreeMarker Template file types. By default, those are `*.ft`, `*.vm` and `*.vsl` for Velocity and `*.ftl` for FreeMarker.

To get coding assistance also for the languages in which the static part of the template is written (those are referred to as template data languages), you should do one of the following:

– Associate the template data languages with files and folders in your project. In this way you specify where in your project you need coding assistance for the corresponding template data languages.
– Add your own extension pattern for the Velocity Template or FreeMarker Template file type and associate that extension pattern with the necessary template data language.

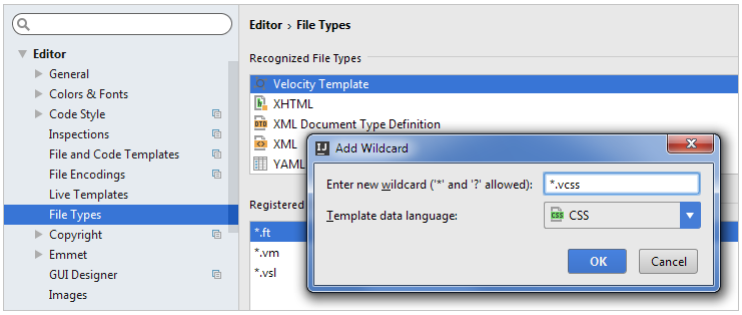
Associating template data languages with files and folders

1. Open the Settings / Preferences dialog (e.g. `Ctrl+Alt+S`).
2. Go to the Template Data Languages page: Languages and Frameworks | Template Data Languages.
3. Click the Template data language cell to the right of the project, or the corresponding directory or file, and select the language.
4. Click OK in the Settings / Preferences dialog.

Individual template files can be assigned a template data language right in the editor. There is the Change template data language to context menu command for that.

Creating an extension pattern for a template data language

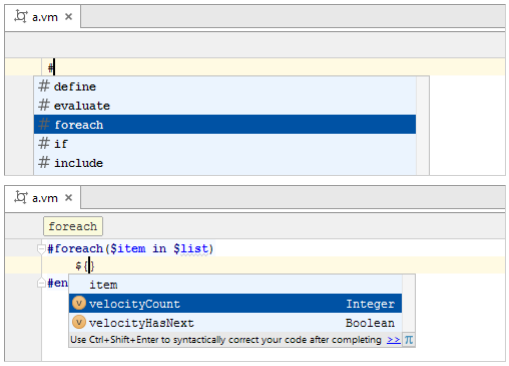
1. Open the Settings / Preferences dialog (e.g. `Ctrl+Alt+S`).
2. Go to the File Types page: Editor | File Types.
3. Under Recognized File Types, select FreeMarker Template or Velocity Template.
4. In the Registered Patterns section, click +.
5. In the Add Wildcard dialog that opens, specify the file name extension pattern and select the language.



6. Click OK in the Settings / Preferences dialog.

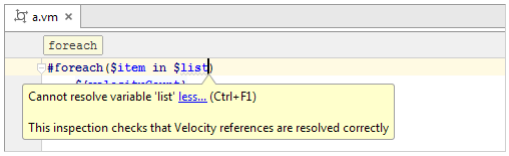
Using code completion

IntelliJ IDEA provides code completion for the template language elements such as directives, variables, built-ins, etc.

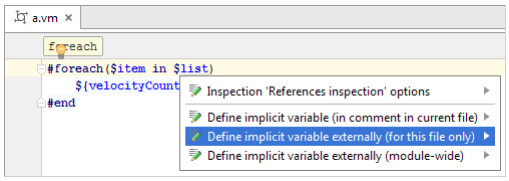


Fixing unresolved references

IntelliJ IDEA provides inspections for detecting unresolved references.



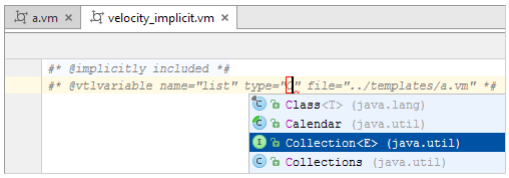
The unresolved references can be fixed by means of intention actions . You can select to add a comment in the same file, or to create a separate file with comments. (Comments in such cases are used to provide missing info about the references.)



In the latter case, a file with the default name `velocity_implicit.vm` or `freemarker_implicit.ftl` is created. The file starts with the comment

```
#* @implicitly included *#
```

To define reference types, code completion is available.



At a later time, you can rename the file, or move it to a different location within the source root, and the reference definitions will not be lost.

Testing Frameworks

IntelliJ IDEA enables usage of the following testing frameworks:

- JUnit
- TestNG
- Groovy JUnit

- FlexUnit
- PHPUnit
- MXUnit
- Cucumber

Testing frameworks support

For each of the supported testing frameworks, IntelliJ IDEA provides:

- Code completion , aware of the specific testing framework.
- Run/debug configurations .
- Ability to create tests .
- Ability to navigate between tests and test subjects .
- Ability to run tests from within the IDE, and view test results in the test runner UI . The test results are shown on the Test Runner tab of the Run tool window .
- Ability to run all tests or features in a directory, specific test classes, test cases or features, individual test methods or examples.
- Code inspections.

Refer to the section Testing for the detailed description of the common testing procedures.

For framework-specific usage guidelines, refer to:

- Testing ActionScript and Flex Applications (FlexUnit)
- Testing PHP Applications (PHPUnit)
- Unit Testing JavaScript
- Creating TestNG Test Classes
- Cucumber

Creating TestNG Test Classes

To create a TestNG test class, you can use the Create Test intention action .

You can also create a new class for your TestNG test and then add the necessary code to that class. This way of creating a TestNG class is described on this page.

Creating a TestNG test class

1. In the Project tool window , right click the directory where you want to create a new test class.
2. Select New | Java Class from the context menu.
3. In the Create New Class dialog, specify the class name and click OK.
4. In the editor, write the code for your test class. Use the TestNG annotations where necessary. For example, you may want to annotate the whole class or individual methods:

```
@Test ()
                    public class NewClassForTesting {}
                    @DataProvider
                    public Object[][] data(){
                    return new String[][]{
                    new String[]{"data1"}, new String[]{"data2"}
                    };
                    }
                    @Test(dataProvider = "data")
                    public void test(String
                    d ){
                    Assert.assertEquals("First
                    Line
                    \n
                    Second Line",
                    "Third Line
                    \n
                    Fourth Line");}
```

If the annotation you add lacks the import statement, the corresponding intention action will suggest you to create one:

Cucumber

Cucumber supports BDD , and enables using features and scenarios written in a human-readable language.

In this section:

- Cucumber
  - Prerequisites
  - Cucumber support
- Enabling Cucumber Support in Project
- Creating .Feature Files
- Creating Step Definition
- Navigating from .Feature File to Step Definition
- Creating Examples Table in Scenario Outline
- Running Cucumber Tests

Prerequisites

Cucumber becomes available in IntelliJ IDEA, provided that the following prerequisites are met:

- Cucumber for Java:
  - Cucumber for Java bundled plugin is enabled.
  - `cucumber-java` version 1.xx is specified as a dependency.
- Cucumber for Groovy:
  - Cucumber for Groovy bundled plugin is enabled.
  - `cucumber-groovy` version 1.xx is specified as a dependency.

Cucumber support

IntelliJ IDEA supports Cucumber for Java and Cucumber for Groovy.

Cucumber feature files are marked with icon.

Cucumber support includes:

- Syntax and error highlighting.
- Quick fix to create step definitions .
- Navigation between step definitions and steps .
- Possibility to describe steps in English or any other language specified in the `# language:` comment.
- Run/debug configuration .
- Possibility to run all features in a directory, a feature, or a single step within a feature.

Enabling Cucumber Support in Project

In this section:

- Prerequisite
- Two ways of creating Cucumber dependencies
  - Using Maven dependencies
  - Using libraries

Prerequisite

Make sure that Cucumber for Java and/or Cucumber for Groovy bundled plugins are enabled. The choice of plugin depends on the type of step definitions to be generated.

Two ways of creating Cucumber dependencies

Cucumber dependencies can be defined in two different ways:

- Using Maven dependencies
- Using libraries

Using Maven dependencies

To enable Cucumber in a Maven project, follow these general steps

1. Create a project from scratch , and select module type Maven. Alternatively, just add a Maven module .
   Do not create a Maven module from archetype!
2. Open for editing `pom.xml` file of the new project or module.
3. In the `pom.xml` , create and expand the section `dependencies` , and add the following code:

```
<dependency>
                    <groupId>info.cukes</groupId>
                    <artifactId>cucumber-java</artifactId>
                    <scope>test</scope>
                    <version>1.0.11</version>
                    </dependency>
                    <dependency>
                    <groupId>info.cukes</groupId>
                    <artifactId>cucumber-jvm</artifactId>
                    <version>1.0.11</version>
                    <type>pom</type>
                    </dependency>
```

4. Run the phase `package` to include all dependencies.

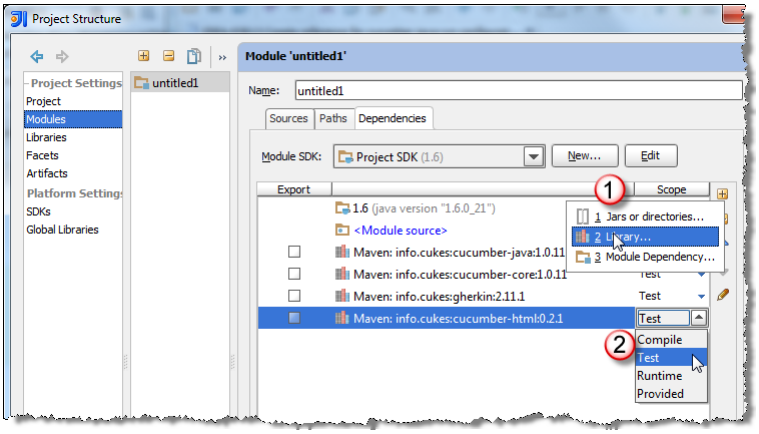
Using libraries

To enable Cucumber support via libraries, follow these general steps

1. In a project where Cucumber tests should be created, click . Project Structure dialog box opens.
2. In the Modules page , click Dependencies tab, and add the required Maven dependencies:
   1. Click +, and choose Libraries from the submenu.
      If the libraries already exists locally, then in the Choose Libraries dialog box, select the desired version of the library `cucumber-java` (for Java) or `cucumber-groovy` (for groovy), and click the button Add Selected.

      If the library does not yet exist locally, click New Library, and select library type From Maven. Then, in the Download Library from Maven Repository dialog box, specify the Maven coordinates (if known), or a keyword, select the target for downloading, and click OK.

      Configure the new library as required (for example, attach documentation URL or sources).
   2. In the Scopes column, click the arrow to reveal the drop-down list, and select Tests.



3. Apply changes and close the Project Structure dialog box.

Creating .feature Files

Prerequisite

Cucumber for Java should be downloaded and installed on your computer!

Once Cucumber for Java support is enabled , you can create feature files .

Creating feature files

To create a feature file

1. In the Project tool window, right-click a directory, where feature files should be created.
2. On the context menu of the target directory, choose New | File, and in the New File dialog box, type `<name>.feature` .
3. In the feature file, type your scenario. Since there are no step definitions, the steps will be highlighted as unresolved.
4. Create step definitions .

Creating Step Definition

On this page:

- Overview
- Creating step definition
- Tips and tricks

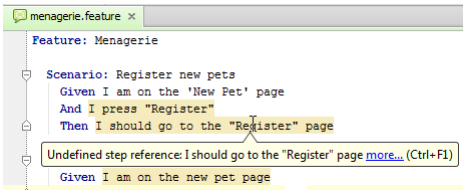
Overview

If a `.feature` file refers to a non-existent step, IntelliJ IDEA's code inspection recognizes and highlights such step, and provides an intention action that helps create missing step definition.

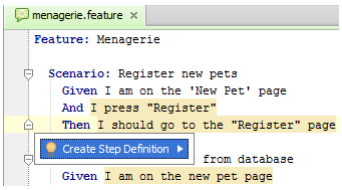
Creating step definition

To create a missing step definition

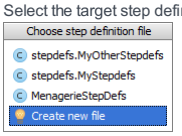
1. While editing the `.feature` file, type a reference to a step definition. IntelliJ IDEA highlights step as undefined, and gives detailed information at the tooltip:



2. Press `Alt+Enter` to show the Create Step Definition intention action:



3. Select the target step definition file from the pop-up list:



You can either select one of the existing step definition files from the suggestion list, or create a new one.

If you opt to create a new step definition file, specify its name, type (Java or Groovy), and the parent directory.

IntelliJ IDEA creates a step definition stub in the specified location.

4. In the selected step definition file that opens in the editor, enter the desired code.
Note that the editor turns into the template editing mode and displays the first input field highlighted with the red frame.

Type step definition in this frame and press `Enter` or `Tab` to complete input and pass to the next input field, where you have to enter your source code. After completing input, the caret moves to the end of the suggested step definition, and the editor returns to the regular mode of operation.

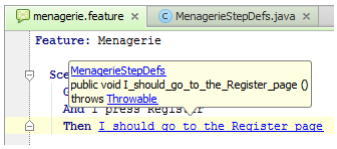
Tips and tricks

- While typing, use code completion `Ctrl+Space` after keywords. Note that you can narrow down the suggestion list by typing characters contained anywhere within a description. So doing, on top of the suggestion list there will be the variants that contain specified characters as prefixes, followed by the variants with the arbitrary occurrences of characters:
- You can find usages of a step definition. To do that, place the caret at the desired definition, and press `Alt+F7`. Refer to the section Finding Usages in Project for details.
- IntelliJ IDEA keeps an eye on the uniqueness of the step definitions. Step definitions with the same names are highlighted.

To navigate from a .feature file to step definition

1. Open the desired `.feature` file in the editor.

2. Do one of the following:
   - Keeping the `Ctrl` button pressed, hover your mouse pointer over a step. The step turns to a hyperlink, and its reference information is displayed at the tooltip:

     Click the hyperlink. The step definition file opens in the editor, with the caret resting at the desired step definition.
   - On the main menu, choose Navigate | Declaration.
   - Press `Ctrl+B`.

Warning! You cannot navigate to a step definition file, if it resides outside a package under the test root.

Creating Examples Table in Scenario Outline

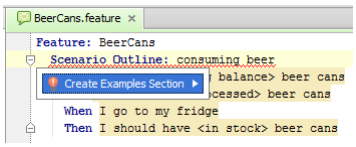
IntelliJ IDEA provides support for scenario outlines, enabling you to describe multiple scenarios by means of templates with placeholders. This support includes:

- Code completion `Ctrl+Space` for keywords.
- Syntax highlighting for keywords, placeholders, and attributes.
- Code inspection to detect missing examples, and a quick fix for generating Examples table stub.

To create Examples table for a scenario outline, follow these general steps

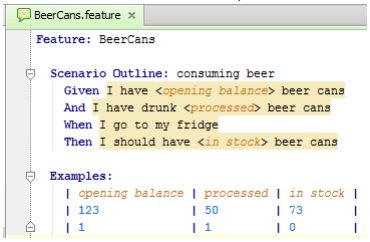
1. Having created a feature file, type the desired scenario outline. Use angle brackets to enclose placeholders. Note that initially the placeholders are not syntactically highlighted. If the steps are not defined, create step definitions . Since the Examples section is missing, IntelliJ IDEA marks Scenario Outline name as an error.

2. Press `Alt+Enter` to show the suggested intention action, and press `Enter` :



   The header row of the Examples table is created; so doing, the placeholders are highlighted both in the table header, and in the scenario outline steps.

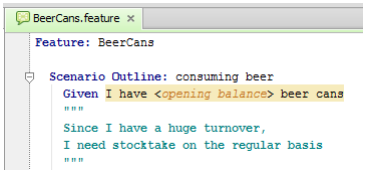
3. Add the desired rows to the Examples table:



   As you add rows, the columns in the Examples section are aligned automatically.

   Note highlighting of the placeholders, and the values in the Examples table, which will be substituted on running examples.

- If a colon is missing after the keyword Examples, it is recognized and marked as a syntax error, and a quick fix suggests to create colon.
- You can add textual notes to the steps of a scenario outline. Such notes should be enclosed in triple quotes; in this case, IntelliJ IDEA perceives the text inside as a string, and displays it when running the scenario step:



Running Cucumber Tests

The Cucumber feature files are run same way as the other executables, with certain run/debug configuration settings.

Prior to running a test, you have to set up run/debug configuration for a particular feature file or scenario, or for the whole bunch of features within a directory. Creating a run/debug configuration is described in the section Creating and Editing Run/Debug Configurations .

Warning! Do not forget to specify the package where step definitions are stored. To do that, in the Glue field of the Edit Run/Debug Configuration dialog box, specify the name of the package in question, or click the browse button to locate it on project.

However, there is always the possibility to run Cucumber tests with the default settings, which is described below.

Results of tests execution are displayed in the Test Runner tab of the Run tool window .

To run all feature files in a directory

1. In the Project tool window , right-click the directory where the feature files are stored.
2. On the context menu of the directory, choose Run all features in <directory name>.

To run a feature

1. In the Project tool window , right-click the desired feature file, or open it in the editor.
2. On the context menu of the feature file, choose Run Feature <name>.

To run a scenario

1. In the Project tool window , right-click the desired feature file, or open it in the editor.
2. On the context menu of the desired scenario, point to Run, and then choose Run Scenario <name>.

TextMate

In this section:

- TextMate
  - Prerequisites
  - TextMate support
- Importing TextMate Bundles
- Editing Files Using TextMate Bundles
- Tutorial: Using TextMate Bundles

Prerequisites

Before you start working with TextMate, make sure that TextMate bundle support plugin is installed and enabled . The plugin is not bundled with IntelliJ IDEA.

Also, make sure that the TextMate bundles are downloaded to your computer.

TextMate support

TextMate files are marked with .

TextMate support in IntelliJ IDEA includes:

- Possibility to import bundles .
- Possibility to establish mappings between the color schemes of IntelliJ IDEA and TextMate.
- Syntax and error highlighting .
- Code completion in `*.markdown` files.

Importing TextMate Bundles

TextMate bundles become available to IntelliJ IDEA when they are downloaded to your computer.

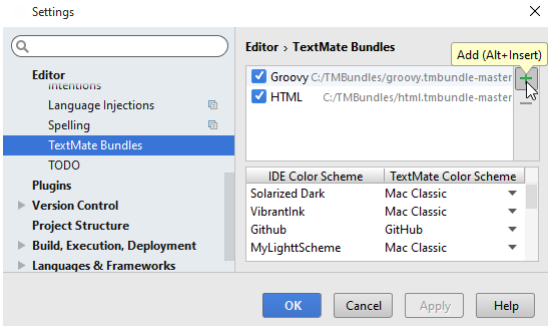
To import a TextMate bundle, follow these general steps

1. Make sure that TextMate bundles you want to import to IntelliJ IDEA, are already downloaded. For example, you can find the desired TextMate bundles on GitHub.

   > **Note** It is important to note that IntelliJ IDEA cannot read the binary bundles. Use only bundle sources instead. You can find them on GitHub.

2. Open Settings/Preferences dialog, and click TextMate Bundles.

3. In the TextMate Bundles page, click +:



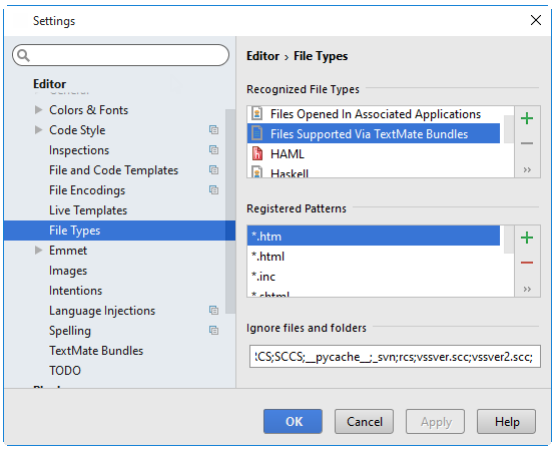
4. In the Select Path dialog box that opens, locate the desired bundle in the file system, and click OK. Repeat this step as required.

5. Apply changes.

6. If necessary, add the desired file type to the to the imported TM bundle.
   For example, if you want Ruby files with `.rb` extension to be supported by the IntelliJ IDEA's TextMate integration, you have to open for editing the file `Ruby.tmbundle\Syntaxes\Ruby.plist`, locate the section `fileTypes`, and under `array` add `rb`.

   Then restart IntelliJ IDEA.

7. Just to make sure that the desired file type will be opened via the TextMate bundles, open File Types page of the Settings/Preferences dialog, among the recognized file types find Files supported via TextMate bundles, and see the list of extensions:



Editing Files Using TextMate Bundles

1. Create a file with the extension `*.markdown` .

   > **Note** A file with the extension `*.markdown` should be associated with the type Files supported via TextMate Bundles, as described in the section Creating and Registering File Types .

2. Open file for editing, and enter the desired text. The lines are highlighted according to the imported bundles.

Tutorial: Using TextMate Bundles

On this page:

– What this tutorial is about
– Prerequisites
– Importing bundles
– Extension conflicts
– Testing

What this tutorial is about

Projects can contain file types unknown to IntelliJ IDEA. While IntelliJ IDEA comes with the built-in support for many programming and scripting languages, you might want to have syntax highlighting for the project-specific languages. For example, a project can contain a shell script, or Perl; a configuration file can exist in a project for the infrastructure automation purposes. If you want to have syntax highlighting for these cases, use the powerful IntelliJ IDEA's integration with the text editor TextMate .

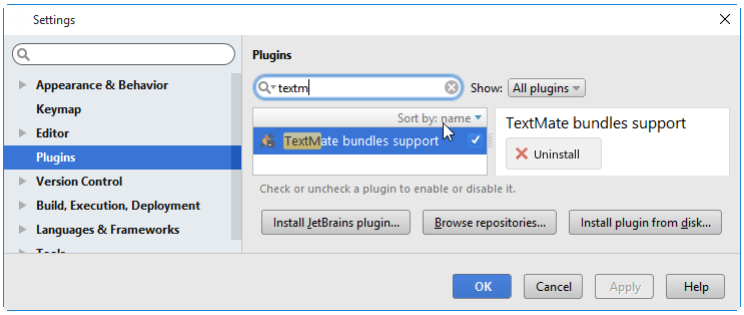
This tutorial aims to walk you step by step through configuring IntelliJ IDEA to use the TextMate Bundles, and editing files with the registered extensions.

Learning TextMate is out of scope of this tutorial. For information about TextMate, refer to the product documentation .

Prerequisites

Make sure that:

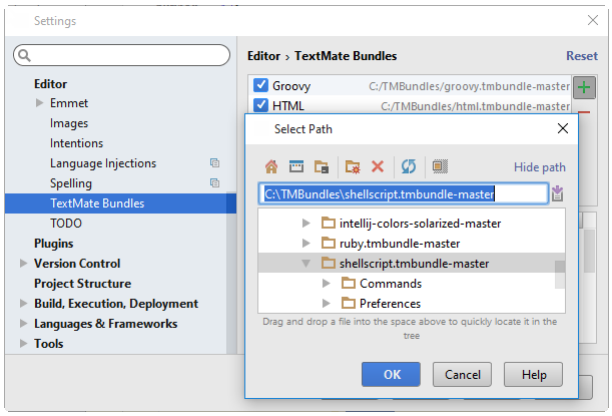
– You have already downloaded bundles you want to use. You can, for example, find the bundles you want to install on GitHub or Subversion .
– You are working with IntelliJ IDEA 13.0 (where TextMate Bundles has been supported) or higher. In this tutorial, IntelliJ IDEA 2016.1 is used.
– Before you start working with TextMate Bundles, make sure that TextMate bundle support plugin is installed and enabled . The plugin is not bundled with IntelliJ IDEA.



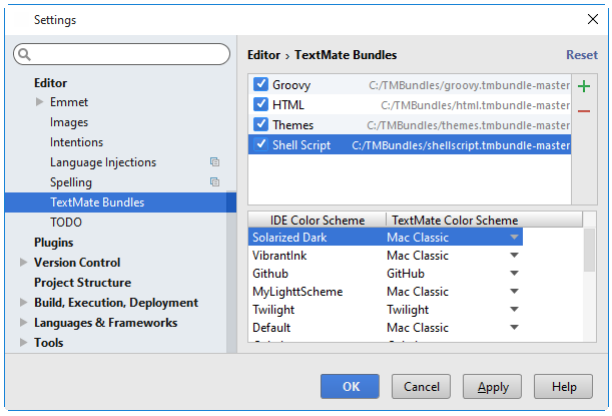
Importing bundles

Suppose you want IntelliJ IDEA to highlight syntax of the Shell Script files. For this purpose, you have already downloaded the Shell Script TextMate Bundle . It now resides on your hard disk, and you only have to import this bundle into IntelliJ IDEA.

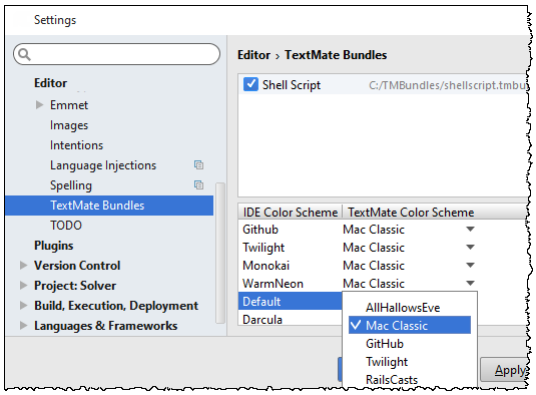
OK, off we go. On the main toolbar, click , and under the Editor node, click TextMate Bundles . Then, in the TextMate Bundles area, click +, and locate the desired bundle on your hard disk:



Click OK to apply changes. The Shell Script bundle appears in the list of recognized bundles, and its local path is visible to the right:



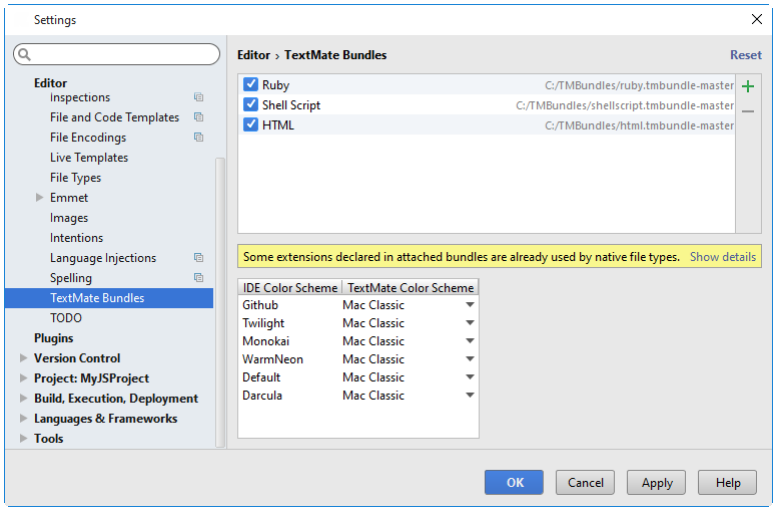
However, we have not yet defined which color scheme IntelliJ IDEA will use to highlight Shell Script syntax. As you already know, IntelliJ IDEA provides a number of color schemes, from the classic-looking ones to the fashionable dark schemes, like Darcula. Have a look at the color scheme mapping section in the lower part of the TextMate Bundles page. By default, the IntelliJ IDEA's Default color scheme maps to the Mac Classic. If we want to use a different color scheme for our bundle, we can click the "Mac Classic" cell in the table of mappings, and select the desired scheme from the list of available ones. However, let's stick to the suggested scheme:



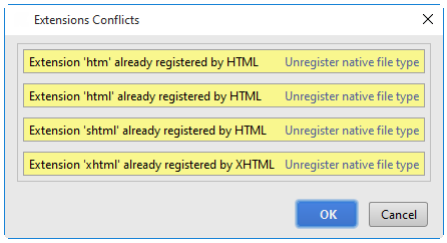
Tip If you want to use a custom TextMate color scheme, you can import a TextMate bundle with schemes, and it will become visible in the list of TextMate color schemes after clicking Apply.

Extension conflicts

Suppose you've imported a bundle that runs into a conflict with the existing file types. The conflict is immediately reported:



Clicking the Show details link opens the dialog box that gives you the chance to unregister the required extensions from the native file types:



The node Files supported via TextMate Bundles now shows the new extensions (`.htm, .html`), and the node HTML lacks these extensions:

Testing

Once a TextMate bundle is added, IntelliJ IDEA provides syntax highlighting for the file types registered with the bundle. Here's a sample script that uses the Shell Script TextMate bundle we've cloned earlier:

Thymeleaf

IntelliJ IDEA supports Thymeleaf up to version 3.

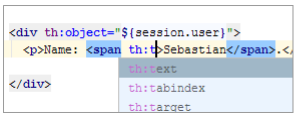
On this page:

- Overview of Thymeleaf support
- Enabling the Thymeleaf plugin
- Thymeleaf support for projects and modules
- Adding Thymeleaf support when creating a project or module
- Adding Thymeleaf support for an existing project or module

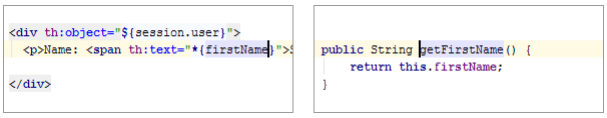
Overview of Thymeleaf support

Thymeleaf support in IntelliJ IDEA includes:

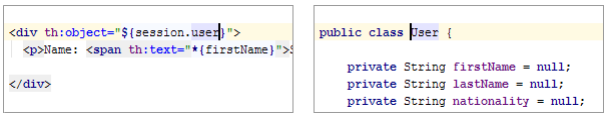
- Code completion for expressions and `th:*` attributes.



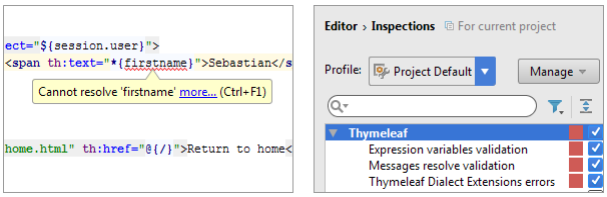
- The Navigate to Declaration feature (Navigate | Declaration or `Ctrl+B`) that lets you jump from a reference in a template to the corresponding getter method, message in a `.properties` file or other appropriate code fragment.



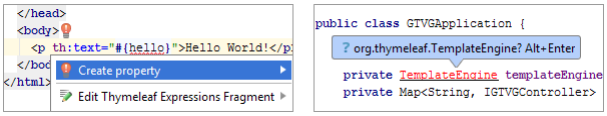
- The Navigate to Type Declaration feature (Navigate | Type Declaration or `Ctrl+Shift+B`) for switching to corresponding type definitions.



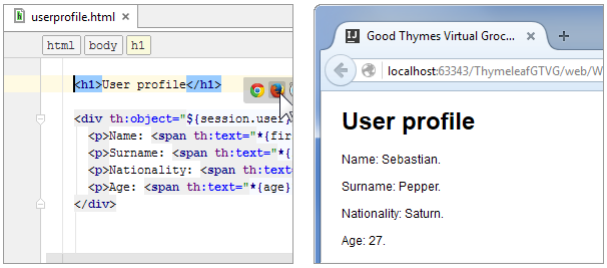
- The Rename refactoring (Refactor | Rename or `Shift+F6`) for referenced properties (and getter methods), iteration and status variables, etc.
- Code inspections that find unresolved references and errors in expression syntax.



- Various Find functions e.g. Find Usages (Edit | Find | Find Usages or `Alt+F7`).
- Intention actions such as Create property for unresolved message references or Import class for adding the `import` statements for `org.thymeleaf.*` classes.



- Preview in a web browser for your prototypes (the static part of your templates) that can be accessed right from the editor.



Enabling the Thymeleaf plugin

To be able to use the Thymeleaf support in IntelliJ IDEA, make sure that the Thymeleaf plugin is enabled. (This plugin is bundled with the IDE and enabled by default.)

See Enabling and Disabling Plugins .

Thymeleaf support for projects and modules

You can add Thymeleaf support when creating a project or module , or for an existing project or module. In all such cases, IntelliJ IDEA downloads the Thymeleaf library files (you can select which of the library files are necessary and which aren't) and adds them to the dependencies of the corresponding module.

You can also create a "Thymeleaf project" by opening an appropriate `pom.xml` file. In that case, the dependencies in your project will be managed by Maven. For more info, see Maven .

Adding Thymeleaf support when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.

As a result, the New Module wizard opens.

2. On the first page of the wizard, in the left-hand pane, select Java . In the right-hand part of the page, specify the JDK that you are going to use.
3. Under Additional Libraries and Frameworks, select the Thymeleaf check box.
   In the lower part of the page, click Configure and select the library files that you want to download in the dialog that opens .
4. You may also want to enable web app development support by selecting the Web Application check box. (For more info, see e.g. Enabling Web Application Support .)
   Click Next.
5. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Adding Thymeleaf support for an existing project or module

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the project or the module folder and select Add Framework Support.
3. In the left-hand pane of the Add Frameworks Support dialog that opens, select the Thymeleaf check box.
4. In the right-hand part of the dialog, click Configure and select the library files that you want to download in the dialog that opens .
5. You may also want to enable web app development support by selecting the Web Application check box. (For more info, see e.g. Enabling Web Application Support .)
6. Click OK in the Add Frameworks Support dialog.

Tiles 3

On this page:

- Plugins
- Libraries
- Overview of Tiles 3 support

Plugins

To be able to use Apache Tiles 3 support in IntelliJ IDEA, make sure that the following plugins are enabled:

- Struts 1.x
- Struts 2 (if you want to use the OGNL support).

(These plugins are bundled with the IDE and enabled by default.)

See Enabling and Disabling Plugins .

Libraries

To get the library files for your Tiles 3 development, use the Apache Tiles Downloads page .

Don't forget to add your libraries to the dependencies of corresponding modules .

Overview of Tiles 3 support

Tiles 3 support in IntelliJ IDEA includes code completion, and syntax and error highlighting for:

- Tiles definition files
- Expression languages (EL): Unified EL and Object-Graph Navigation Language (OGNL)
- Tiles - JSP support tag library

TypeScript Support

In this section:

- TypeScript Support
  - Introduction
  - Preparing for TypeScript development
  - Coding assistance
- Transpiling TypeScript to JavaScript
- Running TypeScript
- Debugging TypeScript
- Using TSLint Code Quality Tool

Introduction

IntelliJ IDEA supports developing and running TypeScript source code. IntelliJ IDEA recognizes `*.ts` files, and allows you to edit them providing full range of coding assistance without any additional steps from your side. TypeScript files are marked with the icon.

To run, debug, and test your code you will need it translated into JavaScript which requires a transpiler.

Preparing for TypeScript development

1. Make sure the JavaScript Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .
2. Make sure the Node.js plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Coding assistance

TypeScript support includes:

- Coding assistance:
  - Code completion for keywords, labels, variables, parameters, and functions.
  - Error and syntax highlighting.
  - Code formatting and folding .
  - Numerous code inspections and quick-fixes .

  This assistance can be provided either by IntelliJ IDEA itself, or based on the data from the built-in TypeScript compiler, or through integration with the TypeScript Language Service . Provision of coding assitance is configured on the TypeScript page of the Settings / Preferences Dialog :
  - Select the Use TypeScript Service check box to get native support from the TypeScript Language Service according to the up-to-date specifications. In this case syntax and error highlighting is performed based on the annotations retrieved from the TypeScript Language Service while code completion lists contain both suggestions from the TypeScript Language Service and suggestions calculated by IntelliJ IDEA itself.
  - Select the Enable TypeScript Compiler check box to active the built-in compiler. When the check box is selected, syntax highlighting and code completion are provided based only on the data from the built-in Typescript compiler. After you select this check box, the fields below on the page become active and you can configure the behaviour of the compiler.
- Refactoring:
  - Common refactoring procedures, such as extract method , inline, rename/move, etc.
  - TypeScript-specific refactoring procedures, such as change signature, extract parameter, extract variable.

  See JavaScript-Specific Refactorings for details.
- Code generation
  - Generating code stubs based on file templates during file creation.
  - Ability to create line and block comments (`Ctrl+Slash` / `Ctrl+Shift+Slash`).
  - Downloading stubs for TypeScript definition files and configuring them as a IntelliJ IDEA JavaScript library (see Configuring JavaScript Libraries).
  - Generating `import` statements for modules, classes, and any other symbol that can be exported and called as a type, see Creating Imports , section Importing TypeScript Symbols.
    You can also configure the type of quotes to be used in `import` statements by choosing Double quotes or Single quotes from the String literal type drop-down list on the Code Style. TypeScript page of the Settings dialog box.
  - Configuring automatic insertion or skipping the public access modifier in generated code.
- Navigation and search through the source code:
  - Navigating with Structure View .
  - Navigate | Declaration (`Ctrl+B`).
  - Navigate | Implementation (`Ctrl+Alt+B`) from overridden method / subclassed class.
- Transpiling to JavaScript for further running, debugging, and testing, see Running TypeScript and Debugging TypeScript .

Transpiling TypeScript to JavaScript

On this page:

- Introduction
- Configuring and activating the built-in TypeScript compiler
- Transpiling TypeScript using the built-in compiler
- Running the built-in TypeScript compiler as a before-launch task
- Transpiling TypeScript using a File Watcher
  - Before you start
  - Installing the TypeScript transpiler globally
  - Installing the TypeScript transpiler in a project
  - Creating a File Watcher
  - Transpiling code

Introduction

TypeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, TypeScript code has to be translated into JavaScript. This operation is referred to as transpilation and the tools that perform it are called transpilers.

In addition to this, transpilation also can involve generation of source maps that set correspondence between lines in your TypeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.

With IntelliJ IDEA, you can have your TypeScript code transpiled into JavaScript in two ways: using the built-in TypeScript compiler or through integration with the typescript tool configured as a file watcher.

Configuring and activating the built-in TypeScript compiler

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Typescript under Languages & Frameworks.
2. On the TypeScript Compiler page that opens, select the Enable TypeScript Compiler check box to activate the compiler. When the check box is selected, syntax highlighting and code completion are provided based only on the data from the built-in Typescript compiler. After you select this check box, the fields below on the page become active and you can configure the behaviour of the compiler.
3. To have the compiler "wake up" upon any change to a TypeScript file, select the Track changes check box.
   When this check box is cleared, the built-in compiler ignores changes to TypeScript files. To re-activate the compiler, open the TypeScript Compiler Tool Window (View | Tool Windows | TypeScript Compiler), and click the Compile All button on the toolbar.

   If you have not opened the TypeScript Compiler Tool Window yet and it is not available from the View menu, choose Help | Find Action, then find and launch the TypeScript Compile All action from the list.

4. In the other fields of the TypeScript Compiler page, specify the following:
   - In the Node interpreter field, specify the location of the Node.js executable file. In most cases, IntelliJ IDEA detects the Node.js executable and fills in the field automatically.
   - In the TypeScript version area, IntelliJ IDEA displays the default bundled version of the built-in compiler.
     To use a custom version of the compiler:
     1. Download the `typescriptServices.js`, `lib.d.ts`, and `lib.es6.d.ts` files from https://github.com/Microsoft/TypeScript/.
     2. Click Edit in the Compiler version area.
     3. In the Configure TypeScript Compiler dialog box that opens, choose Custom directory and specify the folder where the downloaded files are stored.
   - From the Scope drop-down list, choose the scope to apply the compiler in. The available options are:
     - Project Files: all the files within the project content roots (see Content Root and Configuring Content Roots).
     - Project Production Files: all the files within the project content roots excluding test sources.
     - Project Test Files: all the files within the project test source roots.
     - Open Files: all the files that are currently opened in the editor.

     VCS Scopes: these scopes are only available if your project is under version control .
     - Changed Files: all changed files, that is, all files associated with all existing changelists .
     - Default: all the files associated with the changelist `Default`.

     Alternatively, click the Browse button and configure a custom scope in the Scopes dialog box that opens. For more details on scopes, see the pages Scopes and Scopes dialog .
   - Choose the Set options manually to configure transpilation manually:
     - In Command line options field, specify the command line options to be passed to the compiler. See the list of acceptable options at TSC arguments . Note that, the `-w` or `--watch` option (Watch input files) is irrelevant.
     - Select the Generate source maps check box to generate source maps that set correspondence between lines in your TypeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.
     - Select the Compile main file only check box to have IntelliJ IDEA compile only a specific file and the files that are referenced from it and ignore all the other files in the project. This may be helpful if you have a dedicated `main.ts` file which only references other files.
     - Select the Use output path check box to have the built-in compiler store the generated JavaScript files and source map in a custom folder. Specify the path to this folder explicitly or use one of the listed available macros in the format: `$<macros_name>$`. The available macros are:
       - `FileDir`: the path to the folder where the file is stored.
       - `FileRelativeDir`: the path to the file directory relative to the project root.
       - `FileDirRelativeToProjectRoot`: the path to the file directory relative to the module content root the file belongs to.
       - `ModuleFileDir`: the path to the module root folder.
       - `SourcePath`: the path to the source folder under the content root to which the file belongs, see Configuring Content Roots and Configuring Folders Within a Content Root .

- `FileDirRelativeToSourcePath` : the path to the file relative to the source folder under the content root to the file belongs.

- Choose the Use tsconfig.json option, have the built-in compiler analyze the code according to the settings specified in the `tsconfig.json` file .

Transpiling TypeScript using the built-in compiler

1. To activate a previously configured built-in transpiler, select the Enable TypeScript Compiler check box on the TypeScript Compiler page .
   Alternatively, use the buttons on the pane which IntelliJ IDEA displays when you open a TypeScript file.
   - To enable an already configured compiler, click Enable.
   - To configure a compiler, click Configure and specify the settings on the TypeScript Compiler page that opens.
   - To leave the compiler disabled and suppress showing the pane in the future, click Dismiss, whereupon the compiler can be later activated only on the TypeScript Compiler page.

2. View the errors detected during compilation, compile separate files and the entire project in the TypeScript Compiler Tool Window . If you have configured the compiler to track changes, which is the default behaviour, the tool window opens automatically as soon as you start editing a TypeScript file.
   If tracking changes is disabled and you have closed the tool window, to re-open it choose View | Tool WIndows | TypeScript Compiler.
   - Click the Compile Current File toolbar button to run the compiler against the entire file opened in the active editor tab. This approach is helpful if you do not want the compiler to wake up on any change as you edit your code and for this purpose the Track Changes check box on the TypeScript page of the Settings dialog box is cleared.
     If you click this button from the Project Errors pane, after compilation IntelliJ IDEA switches to the Current Errors pane.
   - Click the Compile All toolbar button to run the compiler against all the TypeScript files in the current project. If you click this button from the Current Errors pane, after compilation IntelliJ IDEA switches to the Project Errors pane.
   - To clear the contents of the current pane, click the Clear All toolbar button .
   - To navigate to the fragment of code where an error occurred during compilation, select the error message in question and choose Jump to Source on the context menu of the selection.

The compiler stores the generated output in a separate file. The JavaScript file has the name of the source TypeScript file and the extension `js` , the source map file has the name of the source TypeScript file and the `js.map` extension. In the Project Tree, the files are shown under the source TypeScript file which is now displayed as a node. This presentation is preserved even if you turn off the compiler during a IntelliJ IDEA session.

Running the built-in TypeScript compiler as a before-launch task

You can run the built-in compiler before launching a run configuration to make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files.

1. Check whether the compiler is activated on the TypeScript Compiler page , see Transpiling TypeScript to JavaScript above.
2. Open the Run/Debug Configurations dialog by choosing Run | Edit Configurations on the main menu, and select the required configuration from the list or create it anew by clicking + and choosing the relevant run configuration type.
3. In the dialog box that opens, click + in the Before launch area and choose Compile TypeScript from the drop-down list.
4. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
   - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
   - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.

Transpiling TypeScript using a File Watcher

Using file watchers is recommended if your project consists of several independent parts and you want to transpile them separately. In this case you can configure several file watchers each with its own scope and apply these file watchers independently.

IntelliJ IDEA supports integration with the typescript tool. The TypeScript transpiler translates TypeScript code into JavaScript and creates source maps that set correspondence between lines in your TypeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.

In IntelliJ IDEA, transpiler configurations are called File Watchers. For each supported transpiler, IntelliJ IDEA provides a predefined File Watcher template. Predefined File Watcher templates are available at the IntelliJ IDEA level. To run a transpiler against your project files, you need to create a project-specific File Watcher based on the relevant template, at least, specify the path to the transpiler to use on your machine.

The easiest way to install the TypeScript transpiler is to use the Node Package Manager (npm), which is a part of Node.js . See Installing and Removing External Software Using Node Package Manager for details.

Depending on the desired location of the TypeScript transpiler executable file, choose one of the following methods:

- Install the transpiler globally at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project.
- Install the transpiler in a specific project and thus restrict its use to this project.
- Install the transpiler in a project as a development dependency .

In either installation mode, make sure that the parent folder of the TypeScript transpiler is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

IntelliJ IDEA provides user interface both for global and project installation as well as supports installation through the command line.

Before you start

1. Download and install Node.js . The runtime environment is required for two reasons:
   - The TypeScript transpiler is started through Node.js.
   - NPM, which is a part of the runtime environment, is also the easiest way to download the TypeScript transpiler.

   If you are going to use the command line mode, make sure the path to the parent folder of the Node.js executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the TypeScript transpiler and npm from any folder.
2. Install and enable the NodeJS repository plugin as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

3. Install and enable the File Watchers repository plugin.
   The plugin is not bundled with IntelliJ IDEA, but it is available from the IntelliJ IDEA plugin repository plugin repository. See Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins for details.

Installing the TypeScript transpiler globally

Global installation makes a transpiler available at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project. Moreover, during installation the parent folder of the transpiler is automatically added to the `PATH` variable, which enables you to launch the transpiler from any folder. To install the transpiler globally, do one of the following:

– Run the installation from the command line in the global mode:

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
2. Switch to the directory where NPM is stored or define a `PATH` variable for it so it is available from any folder, see Installing NodeJs .
3. Type the following command at the command line prompt:

```
npm install -g typescript
```

   The `-g` key makes the transpiler run in the global mode. Because the installation is performed through NPM, the TypeScript transpiler is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

   For more details on the NPM operation modes, see npm documentation . For more information about installing the TypeScript transpiler, see https://npmjs.org/package/typescript .

– Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages&Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing the TypeScript transpiler in a project

Installing the transpiler in a specific project restricts its use to this project. To run project installation, do one of the following:

– Run the installation from the command line:

1. Launch the embedded Terminal by hovering your mouse pointer over in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install typescript
```

– Run NPM from IntelliJ IDEA using the Node.js and NPM page of the Settings dialog box.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in template-based projects of the type Node Boilerplate or Node.js Express, which already have the `node_modules` folder. The latter is important because NPM installs the TypeScript transpiler in a `node_modules` folder. If your project already contains such folder, the TypeScript transpiler is installed there.

Projects of other types or empty projects may not have a `node_modules` folder. In this case npm goes upwards in the folder tree and installs the TypeScript transpiler in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be outside your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the TypeScript transpiler is installed there.

In either case, make sure that the parent folder of the TypeScript transpiler is added to the `PATH` variable. This enables you to launch the transpiler from any folder.

Creating a File Watcher

IntelliJ IDEA provides a common procedure and user interface for creating File Watchers of all types. The only difference is in the predefined templates you choose in each case.

1. To start creating a File Watcher, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X on the main menu, and then click File Watchers under the Tools node. The File Watchers page that opens, shows the list of File Watchers that are already configured in the project.
2. Click the Add button + or press `Alt+Insert` and choose the TypeScript predefined template from the pop-up list.
3. In the Program text box, specify the path to the executable file `tsc.cmd` or `tsc` in the `node_modules\.bin` folder.
   – If you installed TypeScript in the current project, the file is stored in the `node_modules\.bin` folder under the project root.
   – In case of global installation, the executable file is stored in the `node_modules\.bin` folder in the user home.

Type the path manually or click the Browse button … and choose the file location in the dialog box that opens.

4. Proceed as described on page Using File Watchers .

Transpiling code

When you open a TypeScript file, IntelliJ IDEA checks whether an applicable file watcher is available in the current project. If such file watcher is configured but disabled, IntelliJ IDEA displays a pop-up window that informs you about the configured file watcher and suggests to enable it.

If an applicable file watcher is configured and enabled in the current project, IntelliJ IDEA starts it automatically upon the event specified in the New Watcher dialog .

- If the Immediate file synchronization check box is selected, the File Watcher is invoked as soon as any changes are made to the source code.
- If the Immediate file synchronization check box is cleared, the File Watcher is started upon save (File | Save All, `Ctrl+S`) or when you move focus from IntelliJ IDEA (upon frame deactivation).

The transpiler stores the generated output in a separate file. The file has the name of the source TypeScript file and the extension `js` or `js.map` depending on the transpiler type. The location of the generated files is defined in the Output paths to refresh text box of the New Watcher dialog . Based on this setting, IntelliJ IDEA detects the transpiler output. However, in the Project Tree, they are shown under the source `.ts` file which is now displayed as a node. This presentation is preserved even if you turn off the file watcher during a IntelliJ IDEA session.

Running TypeScript

On this page:

- Introduction
- Running a file with injected TypeScript from IntelliJ IDEA

Introduction

TypeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, TypeScript code has to be translated into JavaScript. This operation is referred to as transpilation and the tools that perform it are called transpilers.

For more details about using transpilers in IntelliJ IDEA, see the section Using File Watchers .

Running a file with injected TypeScript from IntelliJ IDEA

Note that no run configuration is required for launching applications with injected TypeScript from IntelliJ IDEA.

1. Transpile the TypeScript code into Javascript .
2. In the editor, open the HTML file with the TypeScript reference. This HTML file does not necessarily have to be the one that implements the starting page of the application.
3. Manually replace the reference to the TypeScript file with the reference to the JavaScript file that was generated through the transpilation.
4. Do one of the following:
   - Choose View | Preview file in on the main menu or press `Alt+F2`. Then select the desired browser from the pop-up menu.
   - Hover your mouse pointer over the code to show the browser icons bar: Click the icon that indicates the desired browser.

Debugging TypeScript

TypeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, TypeScript code has to be translated into JavaScript. This operation is referred to as transpilation and the tools that perform it are called transpilers.

Because JavaScript debugging is supported only in Google Chrome and other browsers of the Chrome family, TypeScript can also be debugged only in these browsers.

For more details about using transpilers in IntelliJ IDEA, see the section Using File Watchers .

To debug TypeScript code:

1. Transpile the TypeScript code into Javascript using the built-in compiler or a TypeScript file watcher, see Transpiling TypeScript to JavaScript for details.
2. In the editor, open the HTML files with the TypeScript references.
3. Manually replace the references to the TypeScript files with the references to the JavaScript files that were generated through the transpilation.
4. Configure the JavaScript debugger.
5. Configure and set breakpoints in the TypeScript code. Although actually debugging will be performed against the generated JavaScript, IntelliJ IDEA will emulate TypeScript debugging using the generated source-maps that set correspondence between lines in the TypeScript and in the JavaScript code.
6. Proceed as if you were debugging JavaScript:
    1. Initiate a debugging session. For details about local and remote debugging, about debugging single files and entire applications, see Debugging JavaScript .
    2. If you are using Chrome, install and configure the JetBrains Chrome Extension. For details, see Using JetBrains Chrome Extension .
    3. Step through the program , stop and resume program execution, examine it when suspended , etc.

Using TSLint Code Quality Tool

On this page:

- Introduction
- Before you start
- Installing TSLint
- Activating and configuring the TSLint tool

Introduction

IntelliJ IDEA provides facilities to run TypeScript-specific code quality inspections through integration with the TSLint code verification tool. This tool registers itself as a IntelliJ IDEA code inspection: it checks TypeScript code for most common mistakes and discrepancies without running the application. When the tool is activated, it launches automatically on the edited TypeScript file. Discrepancies are highlighted and reported in pop-up information windows, a pop-up window appears when you hover the mouse pointer over a stripe in the Validation sidebar. You can also press `Alt+Enter` to examine errors and apply suggested quick fixes. Learn more about inspections and intention actions at Code Inspection and Intention Actions .

Before you start

1. The TSLint tool is run through NodeJS, therefore make sure the NodeJS runtime environment is downloaded and installed on your computer. The runtime environment also contains the Node Package Manager(npm) through which TSLint is installed.
2. Integration with NodeJS and NPM is supported through the NodeJS plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Installing TSLint

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the global and at the project level. Click +.
3. In the Available Packages dialog box that opens, select the `tslint` package and click Install Package.

Learn more about installing tools through NPM in Installing and Removing External Software Using Node Package Manager .

Activating and configuring the TSLint tool

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X.
   Expand the Languages&Frameworks node, and then click TSLint under TypeScript.
2. On the TSLint page that opens, select the Enable check box. After that all the controls in the page become available.
3. Specify the location of the Node.js executable file and the path to the tslint package.
4. In the Configuration File area, appoint the configuration to use. By default, IntelliJ IDEA first looks for a `tslint.json` configuration file. IntelliJ IDEA starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. If no `tslint.json` file is found, TSLint uses its default embedded configuration file. Accordingly, you have to define the configuration to apply either in a `tslint.json` configuration file, or in a custom JSON configuration file, or rely on the default embedded configuration.
   - To have IntelliJ IDEA look for a `tslint.json` file, choose the Search for tslint.json option. If no `tslint.json` file is found, the default embedded configuration file will be used.
   - To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the ... button and select the relevant file from the dialog box that opens.
5. If necessary, in the Additional Rules Directory field, specify the location of the files with additional code verification rules. These rules will be applied after the rules from `tslint.json` or the above specified custom configuration file and accordingly will override them.

Vaadin

In this part:

Getting Started with Vaadin

Getting Started with Vaadin-Maven Project

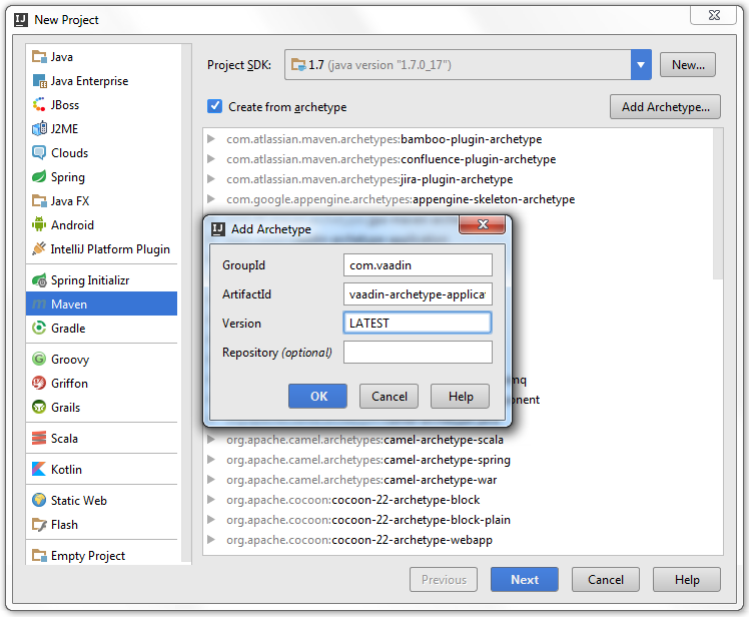
Getting Started with Vaadin-Maven Project

IntelliJ IDEA lets you create and manage a Maven project with Vaadin archetype.

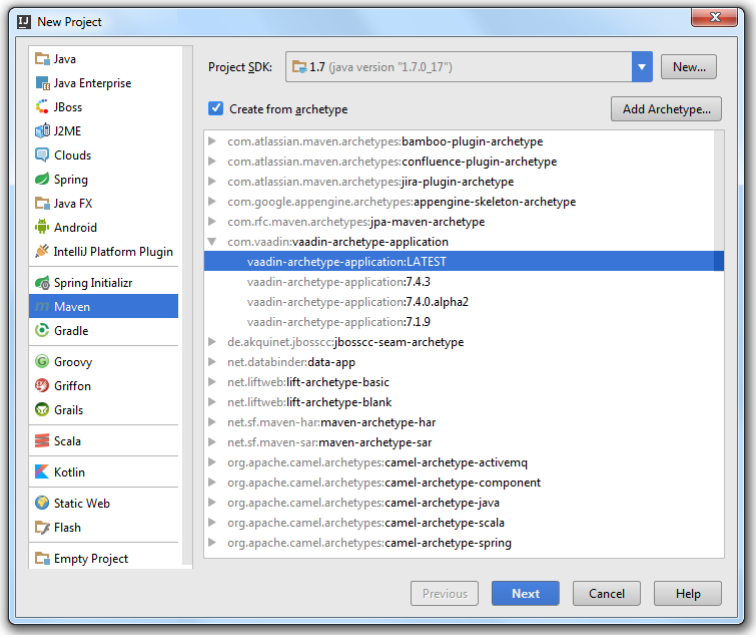
- Creating Maven project with Vaadin archetype
- Import an external Vaadin project with Maven
- Configure run/debug settings and run application

Creating Maven project with Vaadin archetype

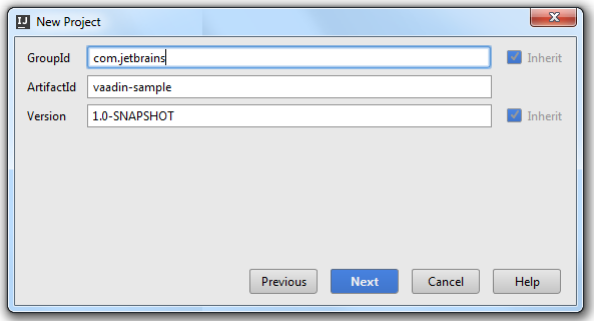
1. Open Project Wizard, under Maven, select Create from archetype check box and click Add archetype.
2. In the dialog that opens, enter maven GroupId, ArtifactId and Version.



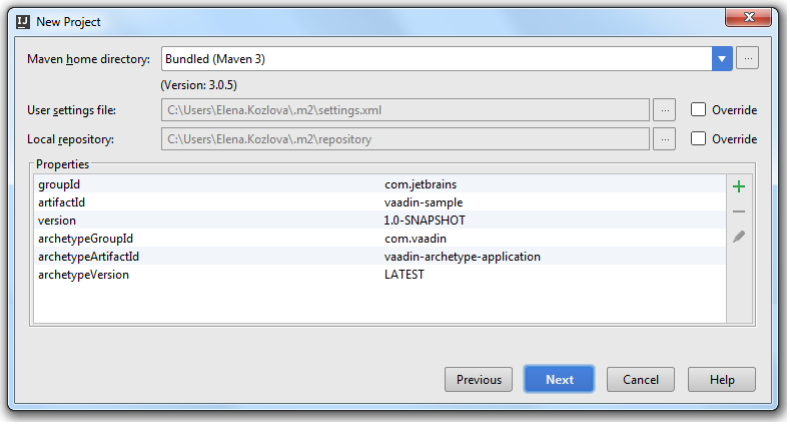
3. After you click OK, the Vaadin archetype is added to the list of available archetypes. Click Next.



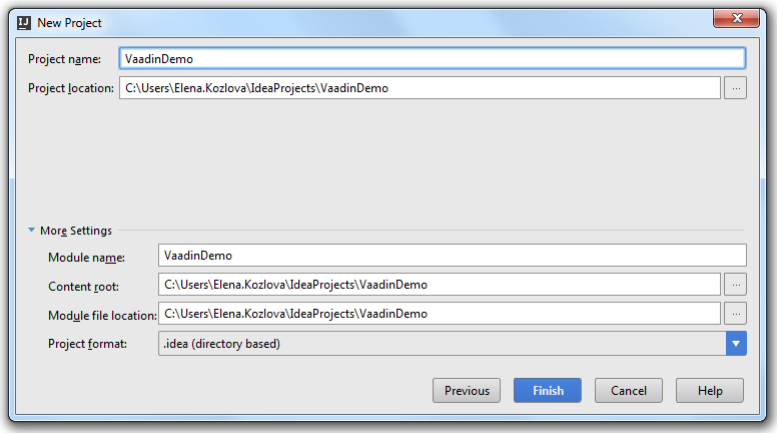
4. On the next page of the wizard, enter maven information for your project and click Next.



5. On the next page of the wizard, enter maven settings for your project and click Next.



6. On the next page of the wizard, enter your project's information and click Finish.

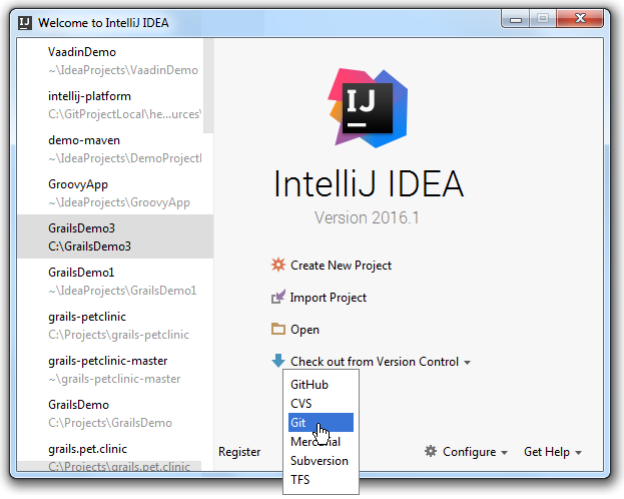


IntelliJ IDEA creates a maven project with pom.xml. IntelliJ IDEA also recognizes that this is a Web application and creates a webapp in your source directory.

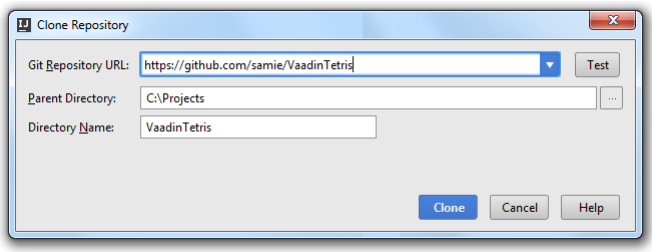
Import an external Vaadin project with Maven

You can import external Vaadin project using VCS and check out your project from, for example, Git. In this case IntelliJ IDEA automatically converts your project structure, detects necessary frameworks and downloads appropriate dependencies.

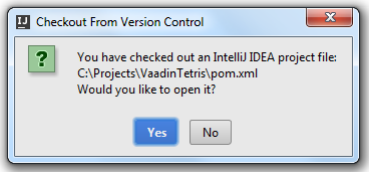
1. In the Project Wizard, click Check out from Version Control and from the drop-down list, select Git.



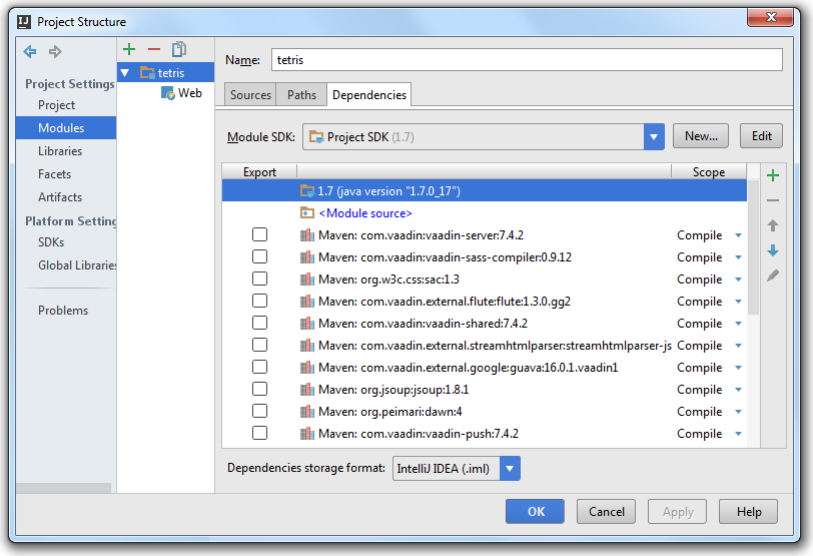
2. Specify Git Repository from which you want to clone your project, a Parent Directory location and the name of the parent directory. Click Clone.



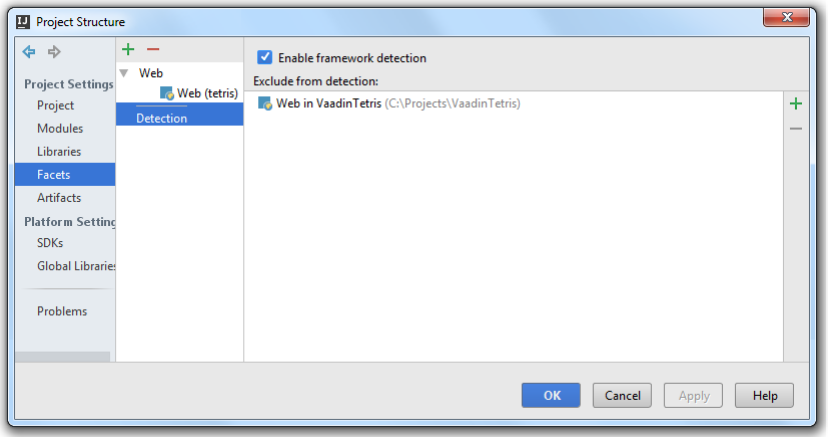
IntelliJ IDEA copies git project to your local directory, recognizes POM file, meaning it recognizes that this is a Maven project with Vaadin framework and imports the project with all the necessary configurations.



3. Check project structure to make sure that all necessary dependencies are downloaded.



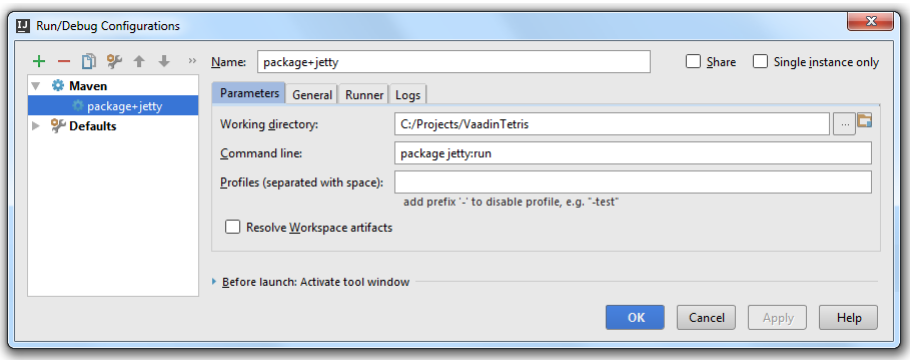
The frameworks are configured automatically, so you can exclude the detected ones.



Configure run/debug settings and run application

IntelliJ IDEA lets you quickly configure Run/Debug settings for your project.

1. On the main menu, select Run | Edit Configurations.
2. Click + to add a new configuration from the list, specify the appropriate parameters for your configuration and press OK.



3. On the main menu press ▶ to run the application.

You can view the output in the default browser http://localhost:8080/ .

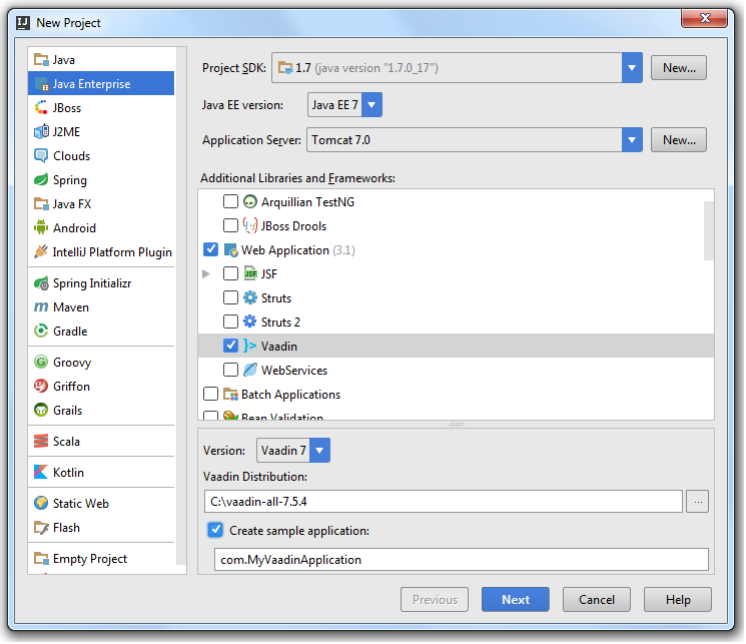
Getting Started with Vaadin

IntelliJ IDEA lets you create and manage projects using Vaadin framework that optimizes your Web application development. Note that before you start creating your project you need to make sure that Vaadin SDK is downloaded on your computer and Vaadin plugin is enabled in IntelliJ IDEA.

- Creating Vaadin project with Project Wizard
- Running Vaadin application
- Debugging Vaadin application

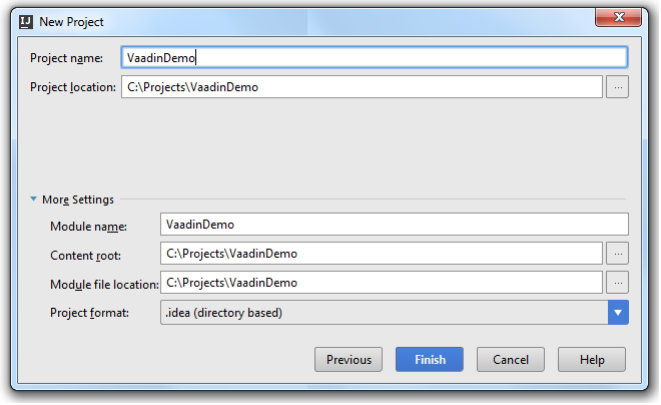
Creating Vaadin project with Project Wizard

1. Open Project Wizard, under Java Enterprise select Vaadin framework.



2. Specify the following settings:
   - Project SDK - specify your project SDK.
   - JavaEE version - specify the version of JavaEE or use the default option.
   - Application Server - specify the application server or use the default server. IntelliJ IDEA creates a run configuration for your Vaadin application.
   - Version - for IntelliJ IDEA 13 and later, you can choose either Vaadin 6 or 7 version.
   - Vaadin distribution - enter the location of Vaadin installation on your machine.

     If Vaadin is not installed, then the warning message with the appropriate link appears. Click the link, download "All-in-One Archive" ZIP file and unpack it to the desired location.
   - Create sample application - select this check box to either enter the name of the sample application or use the default name.
3. Click Next.
4. Specify your project information and click Finish.



IntelliJ IDEA creates a project with Web application and all necessary configurations.

Running Vaadin application

Since the application server is specified you can run your application and view the output in the default browser http://localhost:8080/ .

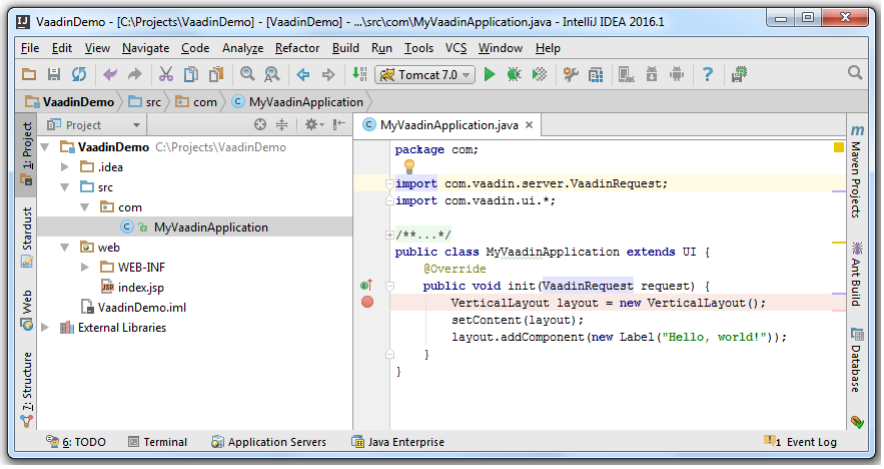
Debugging Vaadin application

IntelliJ IDEA lets you easily start a debugging session for your project.

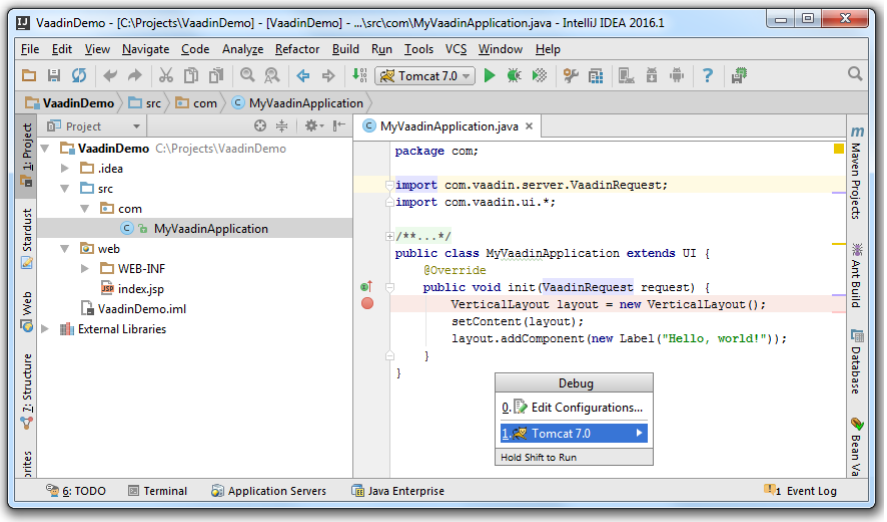
Let's choose one of the debugging actions, for example, Toggle Line Breakpoint.

Place the caret on the desired line of the source code. Press Ctrl+ F8 or from the main menu select Run | Toggle Line Breakpoint

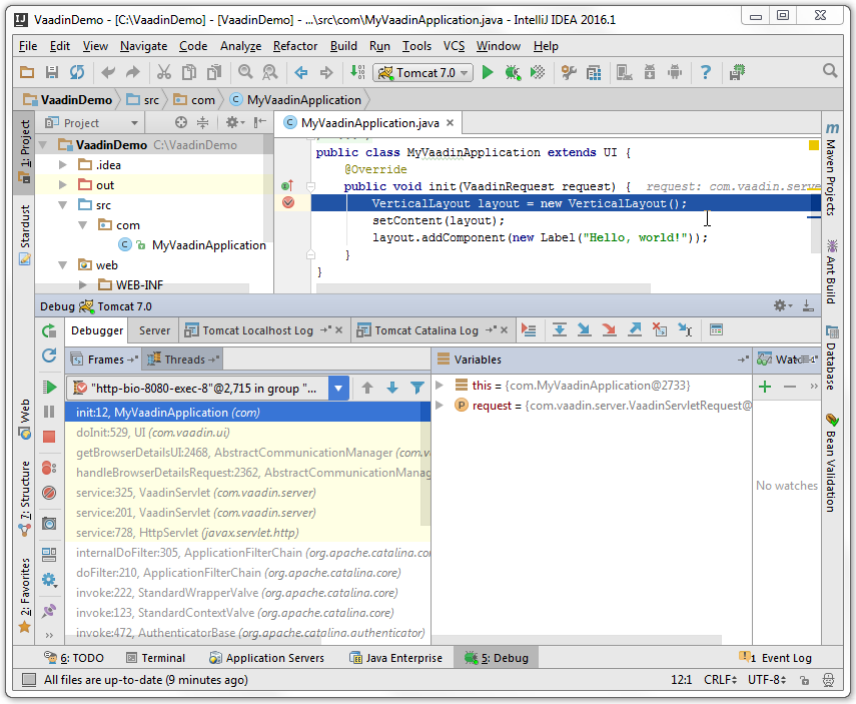
1. Place the caret on the desired line of the source code. Press Ctrl+F8 or from the main menu select Run | Toggle Line Breakpoint.



2. Press Shift + Alt + F9 to start a debugging session.



3. View results in the Debugger console.



Vagrant

In this section:

- Vagrant
  - Prerequisites
  - Basics
  - Vagrant support
  - Preparing to work with Vagrant
  - Initializing the Vagrantfile
  - Creating and launching an instance (virtual machine)
  - Stopping, suspending, resuming, reloading, and destroying an instance (virtual machine)
- Creating and Removing Vagrant Boxes
- Initializing Vagrant Boxes

Prerequisites

Install and enable the Vagrant plugin as described in the sections Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Besides that, make sure that the following prerequisites are met (outside of IntelliJ IDEA):

- Oracle's VirtualBox is installed on your computer.
- Vagrant is installed on your computer, and all the necessary infrastructure is created.
- The parent folders of the following executable files are added to the system PATH variable:
  - `vagrant.bat` or `vagrant` from your Vagrant installation. This should be done automatically by the installer.
  - `VBoxManage.exe` or `VBoxManage` from your Oracle's VirtualBox installation.

Basics

Integration with Vagrant helps you create reproducible development environments defined by VagrantFile configuration files. To have the same environment set up on the machines of all the team members you only need to put the VagrantFile under your team version control. Any team member can set up the required environment by running the `vagrant up` command with the relevant VagrantFile. Vagrant support in IntelliJ IDEA is provided through the Vagrant plugin that comes bundled with IntelliJ IDEA.

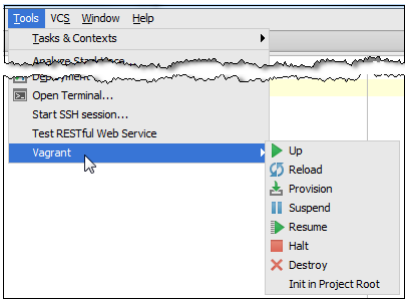
In the context of IntelliJ IDEA, the following terms are used to denote Vagrant-specific notions:

- A Vagrant box in the IntelliJ IDEA user interface or Vagrant base box in the current documentation is a box in the native Vagrant terminology. It denotes a pure image, a skeleton, on the base of which a specific environment is customized, provisioned, and deployed on your machine.
- An instance is a virtual machine. In other words, it is a specific environment customized, provisioned, and deployed on your machine on the base of a Vagrant base box and in accordance with a VagrantFile. In other similar products and documentation, an instance can be referred to as virtual machine.
- An instance folder is the folder where the relevant VagrantFile is stored after initialization and where IntelliJ IDEA will look for it. By default, it is the project root folder.

Vagrant support

Once the Vagrant plugin is enabled, the following changes are made to the IntelliJ IDEA UI:

- A Vagrant page is added to the Settings dialog.
- A Vagrant node is added to the Tools menu. The node contains the following commands that correspond to the standard Vagrant actions:



Vagrant support in IntelliJ IDEA features:

- capability to create new virtual boxes, and delete the unnecessary ones.
- capability to initialize Vagrant boxes , and execute other Vagrant commands without leaving the IDE.
- Multiple Vagrant configuration: when performing the vagrant up command, you are suggested to select the desired virtual box.

Preparing to work with Vagrant

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Vagrant under Tools. The Vagrant page opens.
2. Specify the Vagrant executable file. Because the parent folder of the executable file has been already added to the system PATH variable, type just the name of the executable.
3. In the Instance Folder field, specify the fully qualified path to the directory where the `VagrantFile` is initialized and stored. A `VagrantFile` is a configuration file that defines the instance (virtual machine) you need. The file contains the virtual IP address, port mappings, and the memory size to assign. The file can specify which folders are shared and which third-party software should be installed. According to the `VagrantFile` your instance (virtual machine) is configured, provisioned against the relevant Vagrant base box, and deployed on your computer. A `VagrantFile` is created through the `vagrant init` command.
   When creation of an instance (virtual machine) is invoked either through the `vagrant up` command or through the Tools | Vagrant | Up menu option, IntelliJ IDEA looks

for the `VagrantFile` in the directory specified in the Instance folder field. For more information, see http://docs.vagrantup.com/v2/vagrantfile/ .

You can create a `VagrantFile` in any directory and appoint it as instance folder. If the field is empty, IntelliJ IDEA will treat the project root as the instance folder and look for a `VagrantFile` in it.

4. In the Vagrant Boxes area, configure a list of the predefined Vagrant base boxes available in IntelliJ IDEA. Each item presents a Vagrant base box on which Vagrant configures and launches its instances (virtual machines). The entries of this list correspond to the output of the command `vagrant box list` .
   - To download a new base box, click the Add button +. In the dialog box that opens, specify the URL address to access the base box and the name to refer to it in IntelliJ IDEA. By default, IntelliJ IDEA suggests the URL to the lucid32 box.
     This command corresponds to `vagrant box add <name> <URL>` . As a result, the specified base box is downloaded to your machine.
   - To remove a base box, select it in the list and click the Remove button —. The base box and the nested files are physically deleted from the disk. This command corresponds to `vagrant box remove <name>` .

Initializing the Vagrantfile

A `VagrantFile` is a configuration file that defines the instance (virtual machine) you need. The file contains the virtual IP address, port mappings, and the memory size to assign. The file can specify which folders are shared and which third-party software should be installed. According to the `VagrantFile` your instance (virtual machine) is configured, provisioned against the relevant Vagrant base box, and deployed on your computer. A `VagrantFile` is created through the `vagrant init` command.

You can initialize the Vagrantfile in any folder, just keep in mind that this folder should be specified as the instance folder on the Vagrant page of the Settings dialog box. Otherwise IntelliJ IDEA will be unable to find the relevant VagrantFile during the instance (virtual machine) creation.

To initialize the VagrantFile, do one of the following:

To initialize the Vagrantfile

1. To have the Vagrantfile created in the project root, choose Tools | Vagrant | Init on Project Root on the main menu and select the target project root from the pop-up list that opens. The output of the `init` command is displayed in the Run tool window.
2. To have the Vagrantfile created in a specific folder, launch the embedded Terminal by hovering your mouse pointer over ▣ in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details). Then type the following commands at the prompt:

```
cd <directory to initialize the VagrantFile in>
vagrant init <base box name> <base box url>
```

Once the VagrantFile initialization is successfully completed, you are ready to import the base box, provision and deploy it according to the VagrantFile, thus creating your own instance (virtual machine).

Creating and launching an instance (virtual machine)

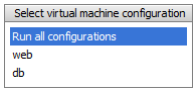
Creating an instance (virtual machine) means importing and provisioning a base box according to the VagrantFile located in the instance folder.

To create an instance, follow these steps:

1. Do one of the following:
   - To have the instance created in the project root, choose Tools | Vagrant | Up.
   - Launch the embedded Terminal by hovering your mouse pointer over ▣ in the lower left corner of IntelliJ IDEA and choosing Terminal from the menu (see Working with Embedded Local Terminal for details).
     Then type the following commands at the prompt:

```
cd <instance folder>
vagrant up
```

2. Select the desired virtual machine configuration from the suggestion list:



Stopping, suspending, resuming, reloading, and destroying an instance (virtual machine)

To reload an instance

- If you have made some changes to the VagrantFile and want a running virtual machine updated in accordance to them, choose Tools | Vagrant | Reload on the main menu or run the following command in the embedded Terminal:

```
vagrant reload
```

For details, see http://docs.vagrantup.com/v2/cli/reload.html .

To suspend an instance

– To temporary stop the operating system on a running virtual machine (guest machine) and save the exact state of the environment as it is at this moment so it can be resumed exactly from this point, choose Tools | Vagrant | Suspend on the main menu or run the following command in the embedded Terminal:

```
vagrant suspend
```

For details, see http://docs.vagrantup.com/v2/cli/suspend.html .

To resume an instance

– To resume a previously suspended operating system on a virtual machine (guest machine) and have it run from the state saved at the suspension moment, choose Tools | Vagrant | Resume on the main menu or run the following command in the embedded Terminal:

```
vagrant resume
```

For details, see http://docs.vagrantup.com/v2/cli/resume.html .

To shut an instance down

– To shut down the operating system on a virtual machine (guest machine), choose Tools | Vagrant | Halt on the main menu or run the following command in the embedded Terminal:

```
vagrant halt
```

For details, see http://docs.vagrantup.com/v2/cli/halt.html .

– To shut down the operating system on a virtual machine (guest machine), stop the virtual machine itself, and remove the resources provisioned on it during the creation (`vagrant up`), choose Tools | Vagrant | Destroy on the main menu or run the following command in the embedded Terminal:

```
vagrant destroy
```

For details, see http://docs.vagrantup.com/v2/cli/destroy.html .

Creating and Removing Vagrant Boxes

It is possible to work with multiple virtual boxes, created in Vagrant. With IntelliJ IDEA, you can add new boxes without leaving the IDE.

The unnecessary boxes can be easily deleted .

To create a Vagrant box

1. Open the Settings dialog box .
2. Under the Project Settings, click Vagrant .
3. Click + button, located next to the Vagrant Boxes table.
4. In the Add Vagrant Box dialog box, specify the name of the new box, and its URL. Then click OK.
   The new box is added to the list of available boxes.

To delete a Vagrant box

1. In the Vagrant page of the project settings, under the Vagrant Boxes table, select the box to be deleted.
2. Click –, and confirm deletion.

The selected box is removed from the list.

Initializing Vagrant Boxes

In this section:

– Basics

– Initializing virtual box in project

– Activating virtual box

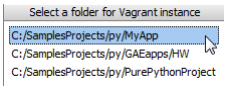
Basics

IntelliJ IDEA makes it possible to execute the `init` procedure on a project root. After performing `init` on a project root, the Vagrant configuration file named `Vagrantfile` is created there. This file can be used for creating a remote interpreter .

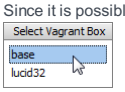
To initialize a Vagrant box in a project root

1. On the main menu, choose Tools | Vagrant | Init in Project Root.

2. In the Select Vagrant Folder pop-up menu, select the project root to be used:



3. Since it is possible to have multiple virtual boxes, specify the particular box you want to work with:



The `vagrant init` command is launched, and shows its output messages in the Run tool window.

If the Vagrant executable is not specified in the Vagrant page, you can still initialize a Vagrant box. However, in this situation, IntelliJ IDEA pops up a file browser, enabling you to select Vagrant executable.

Once `init` is successfully completed, you are ready to perform `vagrant up` command and import to the root the box used at initialization.

To activate a virtual box by executing the 'Vagrant up' command

– On the main menu, choose Tools | Vagrant | Up.
The `vagrant up` command is launched, and shows its output messages in the Run tool window.

Working with Diagrams

In this part:

- Working with Diagrams
  - Prerequisites
  - Basics
  - Features
- Configuring Default Settings for Diagrams
- Viewing Diagram
- Adding Node Elements to Diagram
- Creating Node Elements and Members
- Creating Relationship Links Between Elements
- Deleting Node Elements from Diagram
- Viewing Changes as Diagram
- Viewing Class Hierarchy as a Class Diagram
- Viewing Members in Diagram
- Viewing Siblings and Children
- Viewing Ancestors, Descendants, and Usages
- Editing Module Dependencies on Diagram
- Navigating Through a Diagram Using Structure View

Prerequisites

Before you start working with UML diagrams, make sure that the UML Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Basics

UML modeling support provided by IntelliJ IDEA involves two aspects:

- Reverse engineering, which involves drawing a UML model on the base of the existing code base. Such model helps you get an overview of the classes and packages that comprise your application, relationships between them, explore libraries, and view dependencies.
- Forward engineering, which enables you to design and create a visual model, and populate it with node elements, members and relationships . IntelliJ IDEA automatically generates source code and keeps it synchronized with the model.

UML model in IntelliJ IDEA is represented by a Class diagram in standard notation.

IntelliJ IDEA enables using UML class diagrams to analyze Java, ActionScript/Flex, PHP, and Maven applications, and the structure of the databases and tables. Besides that, you can explore changes committed to VCS.

Features

In IntelliJ IDEA, Class diagram features:

- Ability to view UML model as a diagram in a separate editor tab, in a pop-up window , or as a preview .
- Ability to invoke UML class diagram from the Project, Structure, Database, Maven, Version Control tool windows, the History tab of the Version Control tool window, and Navigation bar.
  In the editor, one can view class diagram for the whole class, or for a symbol at caret.
- Navigation from a diagram element to the underlying source code .
- Highlighting siblings and children classes and packages .
- Refactorings (Rename , Type Migration , Move , Safe Delete , Extract Class , Invert Boolean , Remove Middleman , Inline , Encapsulate , Migrate , Change Signature , Make Static , Convert to Instance Method , Introduce Parameter Object , Wrap Return Value)
- Navigation to class, file or symbol by name and to the last edit location .
- Viewing class or package information at the tooltip, and quick documentation lookup .
- Viewing changed classes as a UML Class diagram.
- Quick hierarchy view in a UML Class diagram pop-up window .
- Viewing subtypes, super classes and classes from signatures .
- Ability to find usages of a node element or member.
- ability to configure default settings for UML Class diagram.

Configuring Default Settings for Diagrams

Default options for the UML Class diagram help define the elements to be shown in diagram, visibility level of the node elements and members, layout, and more. These settings apply to any newly created UML Class diagram. Once a UML Class diagram is open, you can change its display settings as required, using the diagram toolbar and context menu.

Refer to the section Diagrams for the detailed description of controls.

To configure default settings for diagrams

1. Open the IDE Settings , and under the IDE Settings, click Diagrams.
2. In the Diagrams page of the Settings dialog box, choose node you want to configure settings for, and then select the check boxes to define the elements to be shown or hidden in diagram, for example, class members, or visibility level.
3. Select the check boxes that define the Class diagram appearance and behavior: default layout, behavior after layout, and the possibility to view colored links.
4. Apply changes.

Viewing Diagram

On this page:

- Prerequisites
- Basics
- Opening a UML class diagram
- Tips and tricks

Prerequisites

Before you start working with UML diagrams, make sure that the UML Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Basics

You can invoke UML class diagram from different places:

- From the various tool windows.
- From the Navigation bar.
- From the Structure tool window.
- From the editor.

IntelliJ IDEA displays UML diagrams in two modes:

- In a pop-up window.
- In a separate editor tab.

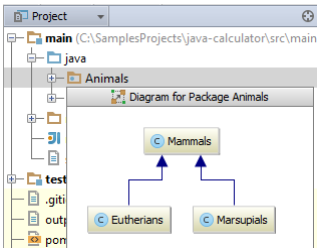
Opening a UML class diagram

1. Select the desired item, or open it in the editor.
2. Do one of the following:
   - On the context menu of the selection, click Diagram, and on the submenu, select the way you want to view the model: Show Diagram or Show Diagram Pop-up.
   - Press `Ctrl+Shift+Alt+U` or `Ctrl+Alt+U`.

The diagram is displayed in the editor tab or in the pop-up window:

Tips and tricks

- You can open a UML class diagram without using your pointing device. Consider such a workflow: press `Alt+Home`, then press `Ctrl+Alt+U`.
- It is possible to view a UML class diagram of a Java package. Just select a Java package in the Project tool window, and press `Ctrl+Alt+U`:



Adding Node Elements to Diagram

You can add the existing node elements to the background of the UML Class diagram, using the context menu action, or drag-and-drop technique.

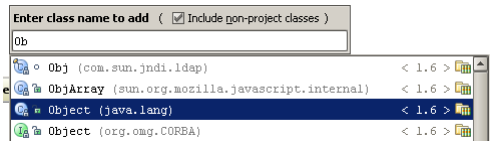
If there are relationships between the node elements in model, and the added element, these relationships will be displayed in diagram.

On this page:

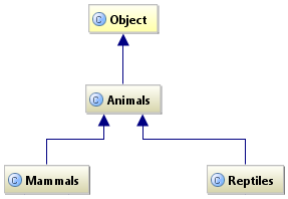
– Adding node elements
– Dragging node elements
– Adding notes

To add a node element to a UML Class diagram

1. Do one of the following:
   – Right-click the diagram background, on the context menu, point to New, and next choose the type of element to be added.
   – Press `Space`.
2. In the Enter class name to add dialog box, start typing the desired class name. IntelliJ IDEA automatically displays suggestion list with matching names. Note that you can include classes outside the scope of your project, by selecting the Include non-project classes check box.

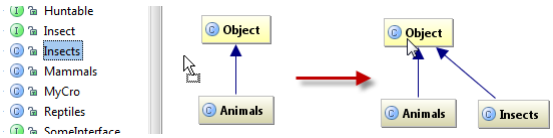


3. Select the desired class from the suggestion list, and press `Enter`.



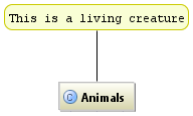
To add a node element using drag-and-drop technique

1. Select one or more desired elements in the Project tool window.
2. Drag selection to the diagram background.



To add a note to a node element

1. Right-click a node element, which you would like to comment.
2. On the context menu, choose New | Note.
3. In the text box, type the desired text. Note that `Enter` starts a new line in the text box. To complete a note, click OK, or press `Ctrl+Enter`.



Creating Node Elements and Members

You can populate your model with node elements (classes, interfaces and enumerations), and delete them from diagram, or from your project. When a node element is created in a UML Class diagram, the corresponding stub file is created in the current package.

In a similar way, IntelliJ IDEA helps create members (fields, methods, or constructors) in the node elements.

It is also possible to clarify the contents of diagrams with notes, which are created same way.

To create a node element in a UML Class diagram

1. Do one of the following:
   - Right-click the diagram background, or a package where a new node element should be created. Next, on the context menu, point to New, and choose element type from the submenu.
   - Press `Alt+Insert`, and choose element type from the pop-up frame.
2. Specify the element name, or type the contents of a note, and click OK.

To create a member in a node element

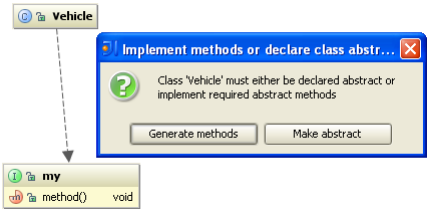
1. Select a node element in a diagram.
2. Do one of the following:
   - On the context menu of the selection, point to New, and then choose the member type on the submenu.
   - Press `Alt+Insert`, and choose member type from the pop-up frame.
3. Depending on the selected member type (field, constant , method , or constructor), specify the member name and the other parameters. See preliminary results in the Preview pane of each dialog.

Creating Relationship Links Between Elements

For creating relationship links between node elements, IntelliJ IDEA provides a special mode, in which drag-and-drop technique can be used for drawing links. As a link is created, the corresponding extends / implements clause is generated in the underlying source code. So doing, the extends link is represented with a solid line, and the implements link is represented with a dotted line.

To create a link between node elements

1. Make sure that is pressed on the diagram toolbar.
2. Draw a link from the source to a target node.
3. Specify whether the target class should be declared abstract, or implement required abstract methods:



To delete a link

1. Select a link between two node.
2. Press `Delete`.
3. In the dialog box that opens, confirm deletion:

Deleting Node Elements from Diagram

IntelliJ IDEA suggests two ways of deleting elements from diagram:

- Remove node elements from view, while leaving the underlying source code intact. This option can be helpful, if you want to explore a certain group of nodes, and remove the unnecessary ones from view.
- Delete node elements from project.

To remove elements from view

1. In the diagram, select one or more elements to be deleted.
2. Press `Delete`.

To delete elements from project

1. In the diagram, select the element to be deleted.
2. Use the Safe Delete refactoring (`Alt+Delete`) to delete selected classes from project.

Viewing Changes as Diagram

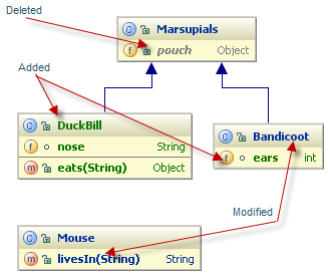
For the modules under version control, you can usually view the list of changed files in the Changes tool window. If you want to evaluate how your changes affect the model, use UML Class diagram. This view presents the complete picture of changes, including relationships between the modified classes.

For this purpose, IntelliJ IDEA provides the action Show changed classes , which is available from the editor, Project tool window, and the Local Changes tab of the Version Control tool window.

Moreover, IntelliJ IDEA helps you analyze how the changes affect a model across revisions, and provides the action Compare all classes from revision on UML .

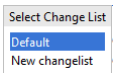
Changes in Class diagram are color-coded:

- Green for added elements.
- Gray for deleted elements.
- Blue for modified elements.



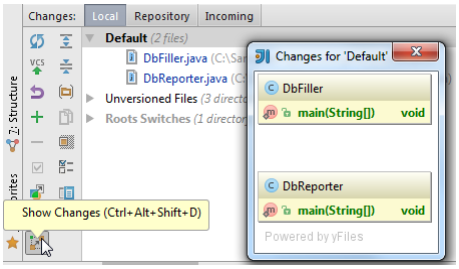
To view changes in UML Class diagram

1. In the Local Changes tab of the Version Control tool window, select the desired changelist from the suggested popup:

   Note that if there is only one changelist, the popup won;t show up.

2. Do one of the following:
   - In the toolbar of the Local tab, click , or press `Ctrl+Shift+Alt+D` .
   - On the context menu of the editor or the Project tool window, choose Diagrams | Show Changes, or press `Ctrl+Shift+Alt+D` .

   The diagram opens in a pop-up window. Double-click a node to view changes in a Differences viewer.

To view changes in revisions as UML Class diagram

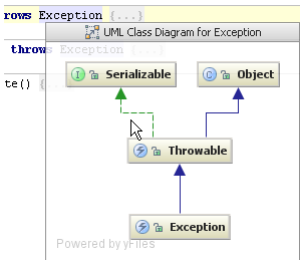
1. In the History tab of the Version Control tool window , select the desired revision.

2. Click , or press `Ctrl+Shift+D` . The diagram opens in a pop-up window.

Viewing Class Hierarchy as a Class Diagram

IntelliJ IDEA helps you view the complete hierarchy of the selected type.

So doing, you can modify the contents of the UML Class diagram, to show ancestor or descendant classes, types used in method signatures, and perform all actions that are available for UML Class diagrams.



To view class hierarchy as a UML Class diagram

1. Open the desired class in the editor, and place the caret at the type you want to see hierarchy for.

2. Do one of the following:
   - On the context menu, point to Diagrams, and then choose Show Diagram, or Show Diagram Popup
   - Press `Ctrl+Shift+Alt+U` or `Ctrl+Alt+U`

Tip You can view type hierarchy for any class selected in the Project tool window:

Viewing Members in Diagram

By default, the diagrams show node elements only. However, you can show members too.

To show members in diagram

1. Press `Ctrl+Shift+Alt+U` to open a diagram.
2. Do one of the following:
   - On the diagram toolbar, click the buttons that correspond to the members you want to show.
   - Right-click the diagram background, and on the context menu, point to the Show Categories command, and then choose the category to be shown or hidden.

Note As you choose a Show Categories command, the state of the corresponding toolbar button changes accordingly.

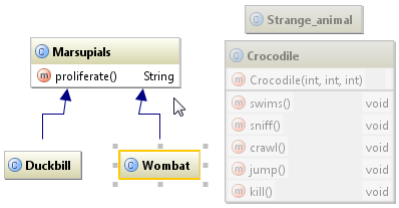
Viewing Siblings and Children

IntelliJ IDEA can smartly tell the nodes that belong to the same package from the extraneous ones.

To highlight sibling nodes

1. Select a node element in diagram.

2. Start navigating through the diagram. So doing, the classes and packages that reside in different packages, are automatically grayed out.

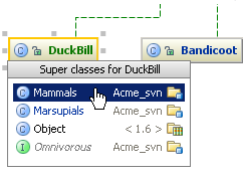


Viewing Ancestors, Descendants, and Usages

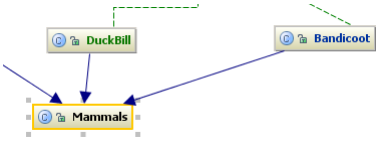
- To show ancestor and descendant types
- To find usages of a diagram element

To show ancestor and descendant types

1. Open UML Class diagram for the desired class or package.
2. Select a node element in diagram.
3. On the context menu, choose one of the following commands, or press keyboard shortcuts:
   - Go to | Implementation `Ctrl+Alt+B`
   - Show Parents `Ctrl+Alt+P`
4. If there are several classes available, select the desired one from the pop-up list:



The selected class will be added to the UML Class diagram, with the corresponding relationship links:



To find usages of a diagram element

1. Select a diagram element.
2. On the context menu of the selection, select the desired Find Usages command.

Editing Module Dependencies on Diagram

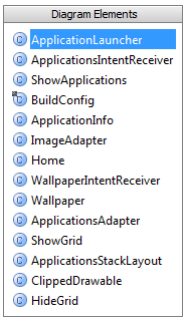
IntelliJ IDEA provides an additional way to explore and change module dependencies using UML class diagram of a module.

To change module dependencies

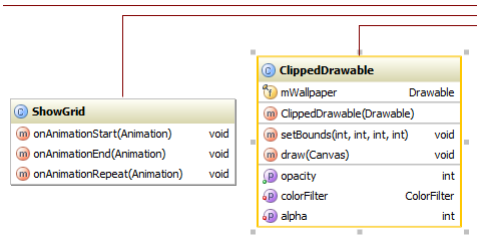
1. Right-click a module node in the Project tool window, or in the Navigation bar.
2. On the context menu, choose Diagrams | Show Diagram.
3. Press `Alt+Insert`.
4. Choose Add Dependency on the pop-up menu.

To navigate through a diagram, follow these general steps

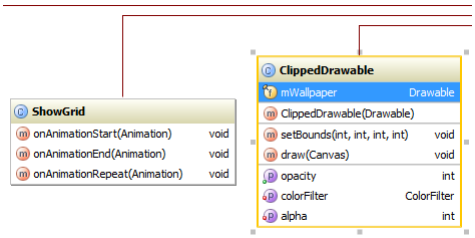
1. Press `Ctrl+F12`. The menu of diagram elements pops up.



2. Use the arrow keys to select the desired diagram element, and press `Enter`. The selected diagram node element becomes active.



3. Press `Enter` once more. The first member of an element gets the focus.



4. The following actions are available:
   - Pressing `Enter` toggles the focus between a diagram element and its members.
   - When a diagram element has the focus, use the arrow keys to jump between elements.
   - When a member has the focus, use the vertical arrow keys to navigate through an element to the desired member.
   - Having selected an element or a member, navigate to the underlying source code .

Web Applications

In this section:

- Web Applications
  - Introduction
  - Developing Web applications
- Enabling Web Application Support
- Populating Web Module
- Configuring Web Application Deployment
- Viewing Pages with Web Contents

Introduction

The term Web application is usually applied to a hierarchical set of folders with HTML, JSP, JavaScript, etc. files and resources, that represent the user interface, and Java classes that implement the application functionality. These folders are arranged according to the standard specified for the server where the application is deployed and executed. To ensure successful deployment and accessibility of your Web application, you need to set up its structure in accordance with the target server standard already at the development stage.

Next, most likely you will need to configure Web application elements, set initialization parameters, and define interaction with EJB, WebServices, etc. You will need various libraries to configure these components and relations and an application descriptor that contains definitions of these components, settings, and constraints.

With IntelliJ IDEA, you can have the basic application structure automatically set up according to the required standard and get extensive coding assistance at all the stages of Web application development.

Developing Web applications

To develop a Web application in IntelliJ IDEA, perform the following general steps:

1. Enable Web application development support to get the necessary libraries, the basic folder hierarchy, and the application descriptor.
2. Populate the Web module : create the required Java classes, configure servlets, filters, listeners, and references.
3. Configure the required static Web content resources .
4. Configure Web application deployment .
5. Deploy and run the application.

Enabling Web Application Support

You can enable Web application development support when creating a project or module . You can as well do that for an existing project or module.

In all such cases, IntelliJ IDEA creates 1) the folder `web` with `index.jsp` intended as a starting page of your application and, optionally, `WEB-INF/web.xml` application deployment descriptor , 2) a Web facet which, among other things, lets you manage your deployment descriptors and 3) an exploded WAR artifact configuration .
Additionally, the Web tool window becomes available for working with the Web part of your application.

- Enabling Web app development support when creating a project or module
- Enabling Web app development support for an existing project or module
- Managing deployment descriptors
- Managing application artifacts

Enabling Web app development support when creating a project or module

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java . In the right-hand part of the page, specify the JDK that you are going to use.
3. Under Additional Libraries and Frameworks, select the Web Application check box.
   Select the version of the Servlet specification to be supported from the Versions list.

   If you want the deployment descriptor `web.xml` file to be created, select the Create web.xml check box.

   Click Next.
4. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

Enabling Web app development support for an existing project or module

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the project or the module folder and select Add Framework Support.
3. In the left-hand pane of the Add Frameworks Support dialog that opens, select the Web Application check box.
4. In the right-hand part of the dialog, select the version of the Servlet specification to be supported from the Versions list.
   If you want the deployment descriptor `web.xml` file to be created, select the Create web.xml check box.
5. Click OK in the Add Frameworks Support dialog.

Managing deployment descriptors

Open your Web facet settings. For corresponding instructions, see Changing Facet Settings .

Your application deployment descriptors are shown in the area under Deployment Descriptors. Use:

- (`Alt+Insert`) to create a new `web.xml` deployment descriptor. (You can do that only if no `web.xml` is currently associated with your application.)
- The Add Application Server specific descriptor button to create an application server-specific deployment descriptor.
- (`Enter`) to replace the selected descriptor file with a different one.
- (`Alt+Delete`) to remove the association of the selected descriptor with your application and, optionally, to physically delete the corresponding file.

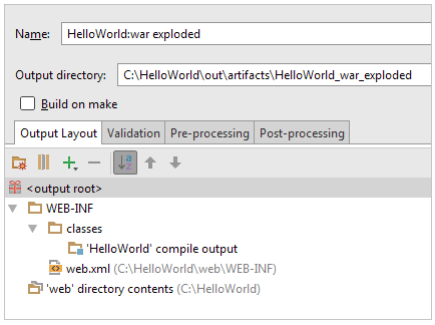
For more information, see Web, EJB and Java EE Application Facet Pages .

Managing application artifacts

To deploy your application to a server, you need an application artifact. For Java Web applications, IntelliJ IDEA provides the following artifact formats:

- Web Application: Exploded. This is a decompressed Web application archive (WAR), a directory structure that is ready for deployment onto a web server.
- Web Application: Archive. This is, obviously, a WAR file.

To manage your artifact configurations, use the Project Structure dialog . If you go there now (e.g. File | Project Structure | Artifacts), you'll see that there is already one exploded WAR artifact configuration created for you by IntelliJ IDEA.



Populating Web Module

After you enable Web development support in a module, IntelliJ IDEA sets up the basic module structure as follows:

- Creates a `web` folder with the `index.jsp` stub file. By default, the `web` folder will be the root of your application after deployment and the `index.jsp` file will be its home page.
- Creates a `WEB-INFO` subfolder that contains the `web.xml` descriptor with an `Action` servlet configured.
- Creates an `src` folder if you specified so when creating your project.

To populate the Web module, perform the following general steps:

- Below the `src` folder, create the Java class files that implement the functionality of your application.
- Configure the static Web content resources that represent the user interface.
- Specify the general application settings .
- Configure the application elements .
- Specify the assembly descriptor references .

Configuring Static Content Resources

Configuring static Web resources involves:

1. Specifying location of the static Web contents on the server .

   After deployment, all the static content resources that implement the user interface should be located below the Web application root directory . By default, IntelliJ IDEA maps the target Web application root directory to the `web` folder which is created after you enable Web development support . The default application home page is mapped to the `index.jsp` stub file which is also created automatically.

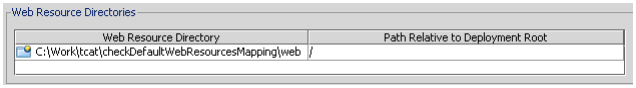
   You can arrange your static content resources in a module in two ways:

   - Store the required resources and directories below the `<project root>\web` folder. After deployment, the entire folder hierarchy of these resources will be copied to the server below the application root.
   - Store the required resources locally wherever you may find suitable and map them to folders on the server.

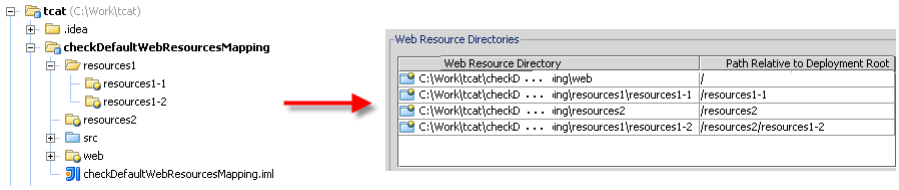
2. Including static Web contents in the artifact to be deployed .

To configure static Web content resources

1. Open the Project Structure settings (`Ctrl+Shift+Alt+S`).
2. Click Modules on the left-hand pane and expand the desired module.
3. Click the Web facet. The right-hand pane of the dialog box displays all the available settings.
4. In the Web Resource Directories area, manage the list of mappings between the local folders with resources you need and the target directories on the server to deploy the resources to. The default mapping between the `<project root>\web` folder and the application rood directory is already on the list:



   - To configure a mapping, click the New button. In the Web Resource Directory Path dialog box that opens specify the desired local folder and the target location relative to the application root folder. By default, IntelliJ IDEA maps the `<project root>\web` folder to the root folder of the application after deployment. For example, if you type a forward slash `/` , the files from the Web resource directory will be copied into the deployment root directory.



   - To edit a mapping, select it in the list, click the Edit button, and update the mapping in the Web Resource Directory Path dialog box that opens.
   - To discard a mapping, select in the list and click the Remove button.

Specifying General Application Settings

Specify the name of the application, provide a brief description, and define in which environment the application can run.

To specify the general application settings

1. Open the `web.xml` application descriptor.
2. Switch to the General tab.
3. In the Display name text box, specify the short display name for the Web application.
4. In the Description text box, type some descriptive text about the application.
5. To enable running the application in a distributed environment, select the Distributable check box.

Alternatively, you can switch to the Text tab and specify all these settings manually.

Creating and Configuring Web Application Elements

The elements and deployment settings of a Web application are defined in the Web module deployment descriptor `web.xml`, which is created automatically when you enable Web application support for a module.

For the sake of consistency with the User Interface, the module deployment descriptors are also referred to as Assembly Descriptors.

You can use the editor and edit the source code using the IntelliJ IDEA coding assistance.

Web modules are populated with elements similarly, according to the common basic procedure . For element-specific details, refer to the relevant topics in this part:

- Creating and Deleting Web Application Elements - General Steps
- Servlets
- Listeners

Creating and Deleting Web Application Elements - General Steps

This topic describes general procedures for elements in a Web module.

To add an element to a Web module

1. Open the Web tool window by choosing View | Tool Windows | Web. The tool window displays all the modules with Web facets.
2. Right-click the desired Web module node and choose New | Servlet/Filter/Listener on the context menu.
3. In the New Servlet/Filter/Listener dialog box that opens, type the name of the element.
4. Specify the package where the element will be created.
5. Specify the class that implements the element.
6. Click OK. A new class is created and opened in the editor.
7. Proceed as described in the relevant element-specific topic.

To remove an element from a Web application

1. In the Web tool window, select the element to be removed.
2. On the context menu, choose Delete and click Yes in the confirmation dialog box.

Servlets

Servlets as a part of a Web application are created and configured through the `<servlet>` and `<servlet-name>` elements in the `web.xml` Web Application deployment descriptor.

In this part:

- Defining the Servlet Element
- Specifying the Servlet Context Parameters
- Specifying the Servlet Initialization Parameters
- Specifying the Servlet Name and the Target Package

Defining the Servlet Element

The Servlet element defines the name of the servlet and specifies the compiled class that implements it. Alternatively, instead of specifying a servlet class, you can specify a JSP.

The Servlet element also contains definitions of initialization attributes .

IntelliJ IDEA provides the facilities to configure , remove , and edit Servlet elements using the IntelliJ IDEA editor.

Note that you can map URL pattern that associates the servlet with the set of URLs that call the servlet using annotations in the editor.

Defining the Servlet element includes:

- Specifying the servlet name and the name of the package where the servlet will be created.
- Customizing the initialization process by specifying the initialization attributes of the servlet.
- Specifying the context-initialization parameters of the servlet

To define a servlet

1. Configure a new servlet .
2. Click OK in the New Servlet dialog box. The new servlet is added to the list in the Servlets Configured pane.
3. Specify the servlet initialization parameters .
4. Specify the servlet context parameters .

To remove a servlet

- Select the servlet in the Servlets Configured pane and click — or select Remove in the context menu.

Specifying the Servlet Context Parameters

Context-initialization parameters enable a servlet to perform during the initialization various one-time activities that impact the entire application.

To define a context parameter

1. Open `web.xml` and switch to the Assembly Descriptor tab.
2. In the Context Parameters pane, click +.
3. Type the name, value, and description of the new context parameter in the relevant text boxes.
4. To remove a parameter, select the relevant row and click —.

Specifying the Servlet Initialization Parameters

You can customize the servlet initialization by specifying initialization parameters of the servlet. This allows the servlet to perform various one-time activities by overriding the `init` method of the Servlet interface. An initialization attribute is defined through the Name and Value parameters. In the Web application deployment descriptor `web.xml`, these parameters are presented as `<param-name>` and `<param-value>` elements.

To specify the initialization parameters

1. Open `web.xml` and switch to the General tab.
2. To add a parameter, click + in the Servlet Initialization Params pane.
3. Specify the name and the value of the initialization attribute in the Name and Value text boxes.
4. Enter a description in the Description text box.
5. To remove a parameter, select the relevant row and click –

To define the servlet name and the target package where the servlet will be created

1. Open the New Servlet dialog box from the Project tool window .
   Alternatively, open `web.xml` and switch to the General tab. Then click + in the Servlets Configured pane.
2. In the <servlet-name> text box, specify the root part of the new servlet name.
3. In the Package text box, specify the name of the target package where the new servlet will be generated. If necessary, use the ... button. The Choose Servlet Package dialog box opens.
4. Select the relevant package or create a new one by clicking . Click OK. You return to the New Servlet dialog box.
5. In the Servlet Class text box, type the name of the class that implements the new servlet. If necessary, use the ... button. The Choose Servlet Class dialog box opens.
6. Select the relevant class and click OK. You return to the New Servlet dialog box.
7. Click OK. The General tab with the Servlet Initialization Params pane displays.
8. Proceed with specifying the initialization attributes and the context-initialization parameters of the servlet.

Listeners

A listener receives notifications on any events related to the Web application, such as deployment/undeployment of the Web application, activation/deactivation of an HTTP session, as well as on adding, removing, and replacing attributes of the application/session.

Listeners, as a part of a Web application, are configured in the Web module deployment descriptor `web.xml`.

To define a listener element

1. Configure a new listener.
2. Switch to the Text view and specify the necessary settings in the Web application deployment descriptor.

Specifying Assembly Descriptor References

Assembly descriptor references define interaction of Web applications with EJB, WebServices, etc. You can specify assembly descriptor references manually in the Web module deployment descriptor `web.xml`.

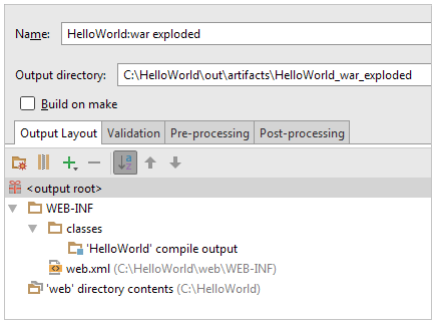
To specify a reference

1. Open `web.xml`.
2. Enter references. You can also edit and remove references in the editor.

Configuring Web Application Deployment

A Web application can be deployed to the server as an exploded directory where files and folders are presented in the file system as separate items or as a Web archive (`WAR` file) which contains all the required files. Therefore you need to configure the layout of your project output so it can be deployed to the server in one of these forms. In IntelliJ IDEA, the layout of a project output is defined through artifacts .

When you enable Web development in a module, IntelliJ IDEA configures an artifact of the type exploded with the following basic structure:



You can use this pre-defined artifact , possibly with necessary customization, or configure a new artifact .

Configuring an artifact to deploy involves:

1. Specifying the artifact type, name, and output directory .
2. Adding static Web content resources .

The suggested deployment configuration procedure reflects the basic workflow which can be flexibly customized depending on your preferences and the requirements to a specific Web application.

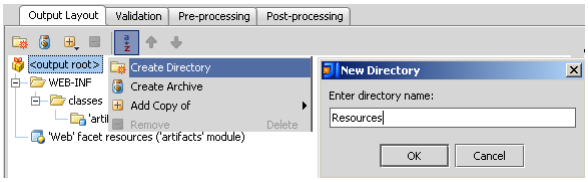
To configure the basic artifact settings

1. Open the Project Structure dialog box (`Ctrl+Shift+Alt+S`).
2. Click Artifacts to open the Artifacts page.
3. Do one of the following:
   - To use a pre-defined exploded directory artifact, select the `<module name>war:exploded` artifact from the list on the left-hand pane. If necessary, change the name and output directory of the artifact in the corresponding fields on the right-hand pane.
   - To create a new artifact, click the New toolbar button on the left-hand pane and choose the artifact type from the New drop-down list.
     - To have the application deployed as a directory, choose Web Application: Exploded.
     - To have the application deployed in the packed form, choose Web Application: Archive.

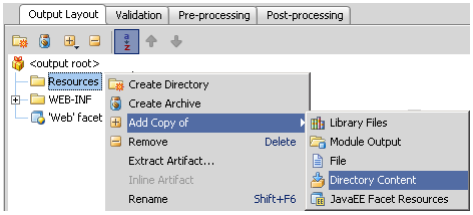
   On the right-hand pane, specify the general settings of the artifact , such as name and output directory, in the corresponding fields.

To add static Web content resources

1. Open the desired artifact and switch to the right-hand pane, the Output Layout tab.
2. With the select the output root node selected, choose the Create Directory item on the context menu or click the Create Directory toolbar button . In the dialog box that opens specify the name of the new folder, for example Resources:



3. With the new folder selected, choose the Add Copy of item on the context menu or click the Add Copy of toolbar button .
4. On the context menu, choose the Directory Content item on the context menu. In the dialog that opens , choose the directory where the required Web content resources are stored.



Viewing Pages with Web Contents

With IntelliJ IDEA, you can perform two opposite operations:

- Preview the output of your Web application in the browser to check whether the pages are rendered correctly. IntelliJ IDEA can display a page preview in a browser of your choice or you can switch between several browsers. IntelliJ IDEA currently supports previews in the following browsers:
  - Mozilla Firefox
  - Internet Explorer
  - Safari
  - Chrome
  - Opera

  No preliminary steps are required to view HTML code. However, to preview PHP pages in the browser, you need to configure synchronization with a server first.
- View the HTML source code of a Web page in the IntelliJ IDEA editor.

In this section:

- Configuring Browsers
- Previewing Pages with Web Contents in a Browser
- Viewing HTML Source Code of a Web Page in the Editor

Configuring Browsers

On this page:

- Integrating browser installations with IntelliJ IDEA
- Choosing the default IntelliJ IDEA browser

Integrating browser installations with IntelliJ IDEA

To make it possible to launch a Web browser from IntelliJ IDEA, you need to integrate installations of Web browsers with IntelliJ IDEA, activate or deactivate launching Web browsers from IntelliJ IDEA and specify whether a browser will be launched by running its executable file or through the default system command .

IntelliJ IDEA is shipped with a predefined list of most popular browsers which you may like to install and use. The items are added to the list in advanced and are not based on the information on actually installed browsers. IntelliJ IDEA presumes that you install browsers according to a standard procedure. Based on this assumption, each browser in this predefined list is assigned an **alias** which stands for the path to its executable file, as IntelliJ IDEA supposes it to be. If in your actual browser installation the path to the executable file is different, you need to specify it explicitly as described below.

In addition to the predefined browsers, you can configure as many custom browser installations as you need using the controls on the toolbar. To create a list of Web browser that can be launched from IntelliJ IDEA:

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Web Browsers under Tools.
2. The Web Browsers page that opens shows a **predefined list of browsers**, possibly extended with previously configured **custom browser installations**. Each browser is presented as a separate table row. The fields in each row show the name of the browser, the family to which the browser belongs, and the path to the browser's executable file or the predefined **alias** that stands for this path.
   - To activate an actually installed browser, select the Active check box next to its name. The browser will be added to the context menu of the Open in Browse menu item and its icon will be displayed in the Browsers pop-up toolbar.
     If the browser was installed according to a standard installation procedure, most likely the **alias** shown in the Path field points at the right location of the executable file. To specify the path explicitly, click … in the Path field and choose the actual location of the executable file in the dialog box that opens.
   - To configure a custom browser installation, click + on the toolbar. In the new row that is added to list, specify the browser name, family and the path to its executable.
   - To change the order of browsers in the list, use the ↑ and ↓ buttons. The order of browsers in the list affect the order in which they will be shown on the context menu of the Open in Browse menu item.
   - To specify a custom profile for **Firefox** or a browser of the **Chrome** family, select the browser in question, click on the toolbar. Depending on the family of the selected browser, the Firefox Settings or Chrome Settings dialog box opens.
     - For **Firefox**, specify the path to the required `profiles.ini` file and choose the profile to use from the drop-down list. Learn more at Firefox browser profile .
     - For **Chrome**, select the Use custom profile directory check box and specify the location of the `chrome-user-data` folder where users' profiles are stored. Learn more about **Chrome** profiles at Multi-profiles .
   - To launch a browser of the **Chrome** with additional options, click on the toolbar and type the required keys in the Command Line Options text box of the Chrome Settings dialog box that opens. Learn more about Chrome command line options by opening chrome://flags in **Chrome**.
   - To remove a browser from the list, select the browser and click — on the toolbar. Note that only custom browser can be removed.

Choosing the default IntelliJ IDEA browser

When you want to preview your application output in the browser by choosing View | Open in Browser on the main menu or Open in Browser on the context menu of a file, you need to choose the browser to open the preview in. You can use a specific browser from the context menu or choose Default Browser. tell IntelliJ IDEA which browser you want to be used by default. This browser is called **IntelliJ IDEA default browser**.

IntelliJ IDEA also opens the **IntelliJ IDEA default browser** to render external resources.

1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Web Browsers under Tools. The Web Browsers page opens.
2. From the Default Browser drop-down list, choose the browser to use by default for previewing pages.
   - To use the default operating system browser, choose System default.
   - To use the browser on top of the list, choose First listed. Change the order or browsers using the ↑ and ↓ icons on the toolbar.
   - To use another browser as default, choose Custom path and specify the location of the executable file of the required browser. Type the path manually or use the Browse button …, if necessary.
3. Specify the way to have the browsers launched.
   - To have a popup window with the enabled browsers appear in the HTML or JSP files, select the Show browser popup in the editor check box.
   - If the Show browser popup in the editor check box is cleared, previewing HTML or JSP files is available only through the View | Open in Browser command on the main menu or the Open in Browser command on the context menu of a file.

Previewing Pages with Web Contents in a Browser

You can preview a file with Web contents in a browser. This can be the IntelliJ IDEA default browser specified in the Web Browsers section of the IDE settings or the one of your choice.

No preliminary steps are required to view HTML code. However, to preview PHP pages in the browser, you need to configure synchronization with a server first.

To preview an opened file in a Web browser, do one of the following:

- On the main menu, choose View | Open in Browser. The current file opens in the default browser.
- With the editor tab having the focus, choose View | Preview file in on the main menu or press `Alt+F2`. Then select the desired browser from the pop-up menu:

- Hover your mouse pointer over the code to show the browser icons bar, and click the icon that indicates the desired browser:

Tip If you don't want to see the icons toolbar, or would like to see the icons of just those browsers you are interested in, clear the Active check boxes for the unnecessary browsers in the Web Browsers section of the IDE settings.

Viewing HTML Source Code of a Web Page in the Editor

You can open the HTML source code of any Web page in the IntelliJ IDEA editor.

No matter which programming language was originally used to develop a page (for example, XML or PHP), IntelliJ IDEA shows the resulting HTML code.

To open the HTML source code of a Web page in the editor

1. Choose File | Open URL.
2. In the Open URL dialog box that opens, specify the URL address of the desired Web page. Type the URL address manually or choose a previously specified one from the drop-down list.

Web Service Clients

In this section:

- Web Service Clients
  - Introduction
  - Developing Web services client applications
- Enabling Web Service Client Development Support
- Monitoring SOAP Messages
- Generating Call to Web Service
- Generating Client-Side XML-Java Binding

Introduction

With IntelliJ IDEA, you can develop the client side for Web services of the following most common types:

- GlassFish/JAXWS2.X RI/Netro 1.X/JWSDP2.0 for developing JAX-WS Web services clients.
- Apache Axis for developing Apache Axis Web services clients.
- JAX-RS for developing RESTful Web Services clients.

You can also enable and manually configure support of the following additional WS engines :

- Apache Axis2
- JAX-RPC
- XFire 1.X / CXF
- JBossWS
- WebSphere 6.X

Developing Web services client applications

To develop a Web services client application, follow these general steps:

1. Create a Java module and enable support of the desired Web services client in it.
2. Generate the client-side XML-Java binding .
3. Modify the generated code to have the client properly initialized.
4. To enable data exchange with the Web service, do one of the following:
   - Declare the variables that will contain the request to the service and the service response.
   - Have IntelliJ IDEA generate the Web service invocation .
5. Populate the module with the necessary classes and methods to implement the user's interface.
6. Run the application.

Enabling Web Service Client Development Support

To develop a Web service client in IntelliJ IDEA, the corresponding module must be relevantly configured and supplied with all the required libraries. This section describes the most common ways to meet these requirements.

To enable Web service client development support, do one of the following:

- Use the dedicated facet. This fast and efficient approach is available for most frequently used WS engines.
  IntelliJ IDEA will download the predefined WS engine implementation version.
- Enable support of an extra WS engine for which the dedicated facet functionality is not provided. This approach is also applicable when you need to use a specific WS engine implementation version.
- Add the relevant libraries to an existing module manually.

Enabling Web Service Client Development Support Through a Dedicated Facet

This approach is applicable if you are going to develop the client side for a Web service of one of the following types:

- GlassFish/JAXWS2.X RI/Netro 1.X/JWSDP2.0.
- Apache Axis.
- RESTful Web Service.

IntelliJ IDEA creates the relevant Web services client module structure and identifies all the necessary libraries automatically through the dedicated Web Services Client facet . Besides, IntelliJ IDEA either locates the previously downloaded required libraries or suggests to download them to the location of your choice.

A Web Services Client facet can be added to the module directly without a parent Web facet.

To enable Web Service Client development support through a dedicated facet, do one of the following

- Create a Java module with the dedicated Web Services Client facet. Besides setting up the relevant module structure and providing all the necessary libraries automatically, IntelliJ IDEA will create a sample package with a `HelloWorldClient` class, which you can use as an example or populate it as necessary to develop your application.
- Add the dedicated facet to an existing module.

In either case, IntelliJ IDEA will download the predefined WS engine implementation version.

To create a module for a Web service client application

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java . In the right-hand part of the page, specify the JDK that you are going to use.
3. Under Additional Libraries and Frameworks, select the WebServices Client check box.
4. If you want sample client code to be generated, select the corresponding check box.
5. Select the desired WS engine implementation from the Version list.
6. You'll need a library that implements the selected WS engine. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library.
     To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the WS engine implementation. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

   Click Next.
7. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

To add the dedicated facets to an existing module

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the module of interest and select Add Framework Support.
3. In the left-hand pane of the Add Frameworks Support dialog that opens, select the WebServices Client check box.
4. If you want sample client code to be generated, select the corresponding check box.
5. Select the desired WS engine implementation from the Version list.
6. You'll need a library that implements the selected WS engine. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library.
     To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the WS engine implementation. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

7. Click OK in the Add Frameworks Support dialog.

Adding WS Libraries to a Web Service Client Module Manually

This approach is applicable if the necessary libraries have been previously downloaded.

To add the necessary libraries to an existing module

1. Open the Module Settings dialog box.
2. With the desired module selected, click the Dependencies tab.
3. Click the Add button again, then select Single-Entry Module Library from the context menu. In the dialog that opens , select the `javaee.jar` library and the required Web service-specific libraries.
   1. The location of the `javaee.jar` library is defined during the installation of IntelliJ IDEA.
   2. The location of the Web-service specific libraries is defined during their download.
4. Click OK when ready.

Enabling an Extra WS Engine (Web Service Client Module)

Besides most common WS engines, support of which is enabled through dedicated facets, with IntelliJ IDEA you can also use the following ones:

- Apache Axis2
- XFire 1.X / CXF
- JBossWS
- WebSphere 6.X

Moreover, you can use any desired WS engine implementation version instead of restricting yourself to the predefined one.

To enable support of an extra WS engine or a specific implementation version

1. Download the desired WS engine implementation.
2. Specify the WS engine location on the Web Services page of the Settings dialog box.
3. On the main menu, choose Tools | Web Services | Enable Web Service Support.
4. In the Enable Web Services Support dialog box that opens specify the following:
   - From the Type drop-down list, select Client.
   - In the Choose Module drop-down list, specify the module to enable Web service client support in.
   - From the Web Service Platform drop-down list, select the desired implementation.
   - To have a sample class generated, select the Add sample code check box.

Monitoring SOAP Messages

While testing the client side of an Apache Axis Web service, it may be helpful to view related SOAP messages which provide procedure details.

Warning! Make sure the target Web service application is running.

To enable monitoring SOAP messages

1. On the main menu, choose Tools | Web Services | Axis | Monitor SOAP Messages.
2. In the Monitor SOAP Messages dialog box that opens specify the Web context of the target application and the port to listen to.

Generating Call to Web Service

IntelliJ IDEA provides coding assistance in developing client calls to Web services, both strictly typed and loosely typed . Coding assistance for calls to strongly typed Web services is Web service interface- and data type- specific and is based on the XML-Java binding generated from the WSDL descriptor.

For will also help you generate calls to loosely typed Apache Axis or XFire Web services. In this case, the basic Web service client coding assistance is provided through the dedicated facet .

To have a call to a strongly typed Web service generated

1. Open the client code in the editor and position the cursor where the call should be generated.
2. Choose Tools | WebServices | WebServices Client Code on the main menu or WebServices | WebServices Client Code on the context menu.
3. From the context menu that opens select the relevant option depending on the type of the target Web service. The available options are:
   - Generate WS Call for Axis
   - Generate WS Call for Axis2
   - Generate WS Call for JAX-WS RI/GlassFish/JWSDP
   - Generate WS Call for XFire
   - Generate WS Call for JAXRPC
   - Generate WS Call for WebSphere
4. Complete the generated invocation code stub by providing the relevant data in the template input fields, using suggestion lists and intention actions, where applicable. The active input field is highlighted with a red frame.
   The contents of the suggestion lists are determined by the Java classes generated from the WSDL descriptor.

To have a call to a loosely typed Web service generated

1. Open the client code in the editor and position the cursor where the call should be generated.
2. Choose Tools | WebServices | WebServices Client Code on the main menu or WebServices | WebServices Client Code on the context menu.
3. From the context menu that opens select the relevant option depending on the type of the target Web service. The available options are:
   - Generate WS Untyped Call for Axis
   - Generate WS Untyped Call for XFire
4. Complete the generated invocation code stub by providing the relevant data in the template input fields, using suggestion lists and intention actions, where applicable. The active input field is highlighted with a red frame.
   The contents of the suggestion lists are determined by the libraries provided through the dedicated facet .

Generating Client-Side XML-Java Binding

To develop well-formed and valid requests from your client to the target Web service, you need to know the available methods of the Web service, the data types it uses, the interface to the service, the acceptable format of requests, the format of generated responses, etc. All this data is presented in the WSDL descriptor of the target Web service.

IntelliJ IDEA can generate the necessary client-side XML-Java bindings based on the desired WSDL descriptor, thus providing you with efficient coding assistance in developing client requests. You only need to specify the URL address of the WSDL descriptor, IntelliJ IDEA will retrieve the necessary data and generate Java classes.

Java code generation is configured in the Generate Java Code from Wsdl or Wadl dialog box, that primarily opens upon enabling the Web service client development support.

To configure generation of the client-side XML-Java binding

1. Open the Generate Java Code from Wsdl or Wadl dialog box by doing one of the following:
   - Create a module and enable support of the Web Services client in it.
   - At any time during the development, select the desired client module in the Project view and choose WebServices | Generate Java Code from Wsdl or Wadl on the context menu.
2. In the Web Service WSDL URL field, specify the URL address of the desired Web service WSDL descriptor.

   Tip If the Status read only field informs you about a WSDL URL connection exception, make sure the target Web service is running and the URL address of its WSDL descriptor is correct.
3. From the Output Path drop-down list, select the directory to place the generated files in.
4. In the Package Prefix drop-down list, specify the package to place the compiled Java classes in.
5. For the client side of an Apache Axis Web service, specify additional configuration options for the code generation process using the following fields:
   - Type Mapping Version
   - Generate TestCase
   - Generate Classes for Schema Arrays
   - Generate Unreferenced Elements
   - Support Wrapped Document/Literal Style

Web Services

In this section:

- Web Services
  - Introduction
  - Developing Web services
- Preparing to Develop a Web Service
- Exposing Code as Web Service
- Managing Deployed Web Services
- Generating WSDL Document from Java Code

Introduction

IntelliJ IDEA supports efficient development, packaging, and deployment of Web services .

The supported standards are:

- Java API for XML Web Services (JAX-WS)
- Apache Axis
- JAX-RS

You can also enable and manually configure support of the following additional WS engines :

- Apache Axis2
- XFire 1.X / CXF
- JBossWS
- WebSphere 6.X

IntelliJ IDEA provides the following development and packaging facilities:

- Setting up the relevant module structure and downloading all the necessary resources based on the dedicated Web services facet you specify.
- Generating the necessary deployment descriptors, mapping and manifest files "on-the-fly".
- Packaging the Web service files, with the EAR file structure following the rules defined by the Enterprise Web Services 1.1 specification.

Tip The Web service development functionality is provided via the Web Services bundled plugin , which is by default enabled. If not, enable it using the Plugins page of the Settings/Preferences dialog box.

Developing Web services

To develop a Web service, follow these general steps:

1. In a Java module , enable support of the relevant Web service.
2. Populate the module with the necessary classes and methods.
3. Compile the developed classes and expose them as a Web service.

   Tip To enable developing the client side of the Web service before the Web service itself is deployed, generate a WSDL document .
4. Configure the artifacts to deploy.

   Warning! Do not forget to include the Web and Web Services facet resources in the artifact.
5. Create a run configuration . On the Deployment tab, create a list of artifacts to be deployed. Specify the application contexts for each of them.
6. Run the application.
7. View and manage deployed Web services in the Deployment Console of the Run tool window.

Preparing to Develop a Web Service

On this page:

- Before You Start
- Basics
- Enabling Web Services Development Support through a Dedicated Facet
- Adding the Necessary Libraries to an Existing Modules
- Enabling an Extra WS Engine

Before You Start

Make sure the Web and JavaEE: WebServices (JAX-WS) bundled plugins are enabled. The plugins are bundled with IntelliJ IDEA and activated by default. If the plugins are not activated, enable them on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Basics

To develop a Web service in IntelliJ IDEA, the corresponding module must be relevantly configured and supplied with all the required libraries and servlet references. This section describes the most common ways to meet these requirements.

Enabling Web Services Development Support through a Dedicated Facet

This approach is applicable if you are going to develop a Web service of the type GlassFish/JAXWS2.X RI/Netro 1.X/JWSDP2.0 or Apache Axis.

Integration with Struts is enabled through the Web and Web Services facets. These dedicated facets contain settings, configuration file paths, and validation rules. This information determines the structure of a module so IntelliJ IDEA detects how to treat the module contents. See Facet for details.

A Web Services facet can be only added as a child of a Web facet . Note that only one Web Services facet is allowed in a module.

You can create a new module with the dedicated facets or add the facets to an existing module. In either case, IntelliJ IDEA downloads the predefined WS engine implementation version.

To create a module with a Web Services facet:

1. Do one of the following:
   - If you are going to create a new project: click Create New Project on the Welcome screen or select File | New | Project.
     As a result, the New Project wizard opens.
   - If you are going to add a module to an existing project: open the project you want to add a module to, and select File | New | Module.
     As a result, the New Module wizard opens.
2. On the first page of the wizard, in the left-hand pane, select Java . In the right-hand part of the page, specify the JDK that you are going to use.
3. Under Additional Libraries and Frameworks, select the Web Application check box.
   Select the version of the Servlet specification to be supported from the Versions list.

   If you want the deployment descriptor `web.xml` file to be created, select the Create web.xml check box.
4. Select the WebServices check box.
5. If you want sample server code to be generated, select the corresponding check box.
6. Select the desired WS engine implementation from the Version list.
7. You'll need a library that implements the selected WS engine. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.
   - Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
     Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

     Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
   - Download. Select this option to download the WS engine implementation. (The downloaded files will be arranged in a library .)
     Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
   - Set up library later. Select this option to postpone setting up the library until a later time.

   Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

   Click Next.
8. Specify the name and location settings. For more information, see Project Name and Location or Module Name and Location .
   Click Finish.

To add a Web Services facet to an existing module:

Adding the Necessary Libraries to an Existing Modules

This approach is applicable if the necessary libraries have been previously downloaded.

1. Open the Module Settings dialog box .
2. With the desired module selected, click the Dependencies tab.
3. To enable Web development, click the Add button and select Single-Entry Module Library from the context menu. In the dialog that opens , select the `javaee.jar` library and click OK.
4. Click the Add button again, then select Single-Entry Module Library from the context menu. In the dialog that opens , select the `javaee.jar` library and the required Web service-specific libraries.
   - The location of the `javaee.jar` library is defined during the installation of IntelliJ IDEA.
   - The location of the Web-service specific libraries is defined during their download.
5. Click OK when ready.

Enabling an Extra WS Engine

Besides most common WS engines, support of which is enabled through dedicated facets, with IntelliJ IDEA you can also use the following ones:

- Apache Axis2
- XFire 1.X / CXF
- JBossWS
- WebSphere 6.X

Moreover, you can use any desired WS engine implementation version instead of restricting yourself to the predefined one.

To use a custom WS engine:

1. Download the desired WS engine implementation.
2. Specify the WS engine location on the Web Services page of the Settings dialog box.
3. Enable Web development support.
4. On the main menu, choose Tools | Web Services | Enable Web Service Support.
5. In the Enable Web Services Support dialog box that opens specify the following:
   - From the Type drop-down list, select Server.
   - In the Choose Module drop-down list, specify the module to enable Web service support in.

     Warning! The drop-down list contains only modules with Web development support enabled.
   - From the Web Service Platform drop-down list, select the desired WS engine.
   - To have a sample class generated, select the Add sample code check box.

Exposing Code as Web Service

Suppose, you have a piece of code that implements certain functionality and you want this functionality to be available on the Web through a specific protocol. In this case you need to transform the existing code into a Web service and deploy. Such transformation is called exposing a class as Web service and should be applied to each class. The action updates the Web service descriptors and generates additional deployment code if needed. This action is required every time any Web service method signature is changed or a new method is added or removed.

To expose a class

1. Position the cursor at the class name in the editor and press `Alt+Enter` or click the yellow bulb icon.
2. From the suggestion list, choose Expose Class as Web Service.
3. In the Expose Class As Web Service dialog box that opens, specify the following:
   - The name and URL address of the Web service.
   - The protocol and encoding style used when accessing the public operations of the Web service.
   - The type information, including name, operations, parameters and data comprising the interface of the Web service.

   IntelliJ IDEA automatically adds the service description to the `server-config.wsdd` file.

Managing Deployed Web Services

IntelliJ IDEA provides the possibility to view which Web services are currently deployed as well as to re-deploy and undeploy Web services without re-starting the application server.

These operations are enabled only when the corresponding server is running.

To get a list of currently deployed Web services, do one of the following

- On the main menu, choose Tools | Web Services | Show Deployed Web Services. The Web services deployed under various application contexts are listed in the browser.
- Open the Deployment Console of the Run tool window. The console shows which of the deployment artifacts are successfully deployed and which are not. Find the artifacts that constitute the Web services in question.

To manage the list of deployed Web services, perform these general steps

- Open the Deployment Console of the Run tool window. The console shows which of the marked for deployment artifacts are successfully deployed and which are not. Find the artifacts that constitute the Web services in question.
- To set the deployment statuses up-to-date, click the Refresh Deployment Status button .
- To undeploy a Web service, select the relevant artifacts in the list and click the Undeploy button .
- To have IntelliJ IDEA deploy all the artifacts marked for deployment, click the Deploy All button .

Generating WSDL Document from Java Code

The available functionality of a Web service, the ports to access them, the acceptable format of requests, the format of generated responses, etc. are reflected in the Web service WSDL descriptor , which is normally generated on the server during the Web service deployment. A major part of Web service client development is implementing generation of requests to the service and parsing responses from it in compliance with the WSDL descriptor settings.

Suppose you have developed a Web service and want its client side development start before the Web service itself is deployed, that is, before the WSDL descriptor is generated on the server. With IntelliJ IDEA, you can have it generated before deployment.

To create a WSDL descriptor from Java code

1. Select the desired class name in the editor.
2. Choose Tools | Web Services | Generate WSDL From Java Code on the main menu or choose Web Services | Generate WSDL From Java Code from the context menu.
3. In the Generate WSDL From Java dialog box that opens specify the following:
   - The name and URL address of the Web service.
   - The protocol and encoding style used when accessing the public operations of the Web service.
   - The type information, including name, operations, parameters and data comprising the interface of the Web service.

   Click OK, when ready.

XML Refactorings

IntelliJ IDEA lets you refactor your XML code.

The XML refactoring support is based on the Refactor-X plugin. This plugin is bundled with the IDE and enabled by default.

To access XML refactoring commands, you can use the main menu or the context menu in the editor: Refactor | XML Refactorings | <Refactoring_Name>.

The following refactorings are available:

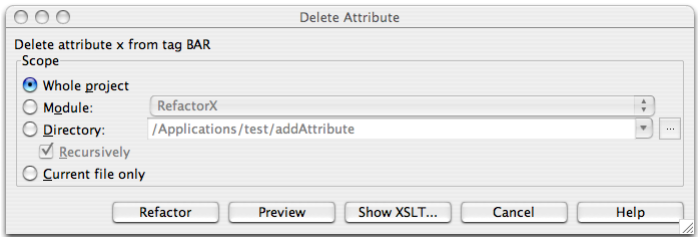
- Delete Attribute
- Replace Attribute with Tag
- Replace Tag with Attribute
- Add Attribute
- Add Subtag
- Move Attribute in
- Move Attribute Out
- Change Attribute Value
- Convert Contents to Attribute
- Expand Tag
- Collapse Tag
- Merge Tags
- Split Tags
- Delete Tag
- Unwrap Tag
- Wrap Tag
- Wrap Tag Contents

Delete Attribute

The Delete Attribute refactoring allows you to delete a set of attribute definitions on a set of XML tag. If this refactoring is invoked, all attributes matching the selected attribute name on tags with the selected tag name may be removed. This bulk removal of attributes may be useful as XML schemas evolve.

Deleting an attribute

1. In the editor, place the cursor withing the attribute to be deleted.
2. Select Refactor | XML Refactorings | Delete Attribute from the main or the context menu.



3. Determine the scope of the deletion. All attributes matching name of the attribute selected on the tag selected will be deleted, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
4. Press Preview button to make IntelliJ IDEA to search for usages of the selected attribute Find window.
5. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Please note the following:

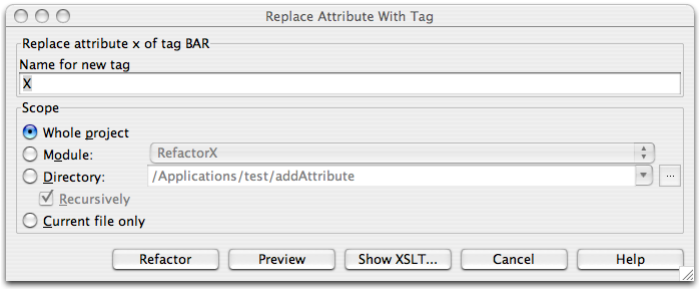
- Pressing Preview opens the Refactoring preview window displaying all found usages of the attributes to be removed. In this window you can exclude/include usages you want to refactor.
- Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.
- The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Replace Attribute With Tag

The Replace Attribute with Tag refactoring allows you to replace attribute definitions on a set of XMLs tag with equivalent sub-tags. If this refactoring is invoked, all attributes matching the selected attribute name on tags with the selected tag name may be removed, and equivalent sub-tags created. This bulk transformation of attributes to tags is useful as XML schemas evolve.

Converting an attribute into a tag

1. In the editor, place the cursor within the attribute to be converted.
2. Select Refactor | XML Refactorings | Replace Attribute with Tag from the main or the context menu.



3. Determine the name of the tags to replace the selected attributes.
4. Determine the scope of the replacement. All attributes matching name of the attribute selected on the tag selected will be replaced, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
5. Press Preview button to make IntelliJ IDEA to search for usages of the selected attribute Find window.
6. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Pressing Preview opens the Refactoring preview window displaying all found usages of the attributes to be replaced. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

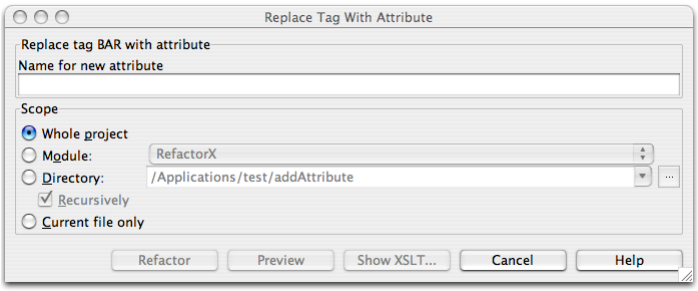
Refactoring preview window may appear anyway, if the files to be affected are read-only.

Replace Tag With Attribute

The Replace Tag with Attribute refactoring allows you to replace sub-tags definitions on a set of XMLs tag with equivalent attributes. If this refactoring is invoked, all tags matching the selected tag's name on tags with the selected parent tag name may be removed, and equivalent attributes created. This bulk transformation of sub-tags to attributes is useful as XML schemas evolve.

Converting a tag into an attribute

1. In the editor, place the cursor within the tag to be converted.
2. Select Refactor | XML Refactorings | Replace Tag with Attribute from the main or the context menu.



3. Determine the name of the attributes to replace the selected tags.
4. Determine the scope of the replacement. All tags matching name of the tag selected on the parent tag selected will be replaced, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
5. Press Preview button to make IntelliJ IDEA to search for usages of the selected tag Find window.
6. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Pressing Preview opens the Refactoring preview window displaying all found usages of the tags to be replaced. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

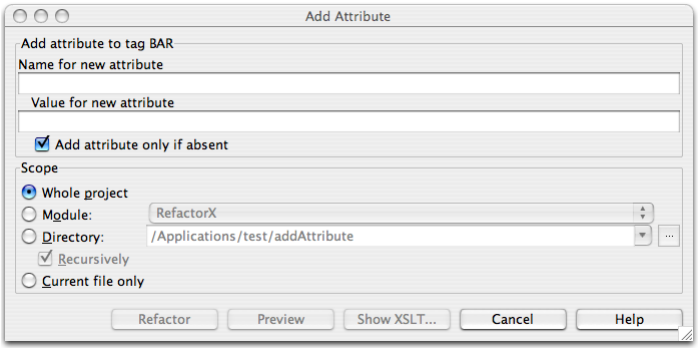
The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Add Attribute

The Add Attribute refactoring allows you to add an attribute to a set of XML tags. If this refactoring is invoked, attributes matching the requested attribute name and value will be added to tags with the selected tag name. Optionally, the requested attribute may be added only to tags that do not already contain the selected attribute. This bulk addition of attributes is useful as XML schemas evolve.

Adding an attribute

1. In the editor, place the cursor within the corresponding tag.
2. Select Refactor | XML Refactorings | Add Attribute from the main or the context menu.



3. Determine the name of the attribute to add to the selected tags.
4. Determine the value of the attribute to add to the selected tags.
5. Determine whether tags that already contain the requested attribute will be ignored, or will have their values changed.
6. Determine the scope of the addition. All tags matching name of the selected will be replaced, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
7. Click Preview to make IntelliJ IDEA search for usages of the selected attribute.
8. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Clicking Preview opens the Refactoring preview window with all found usages of the attributes to be replaced. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window, displaying a small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

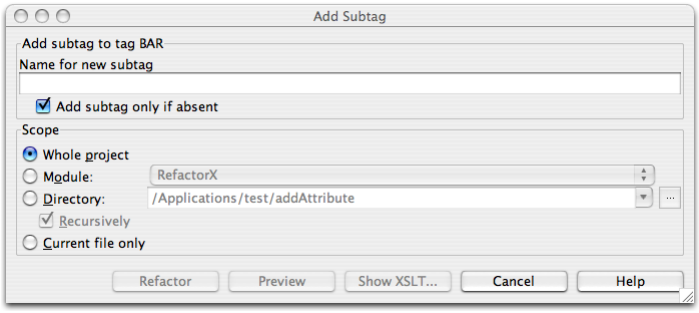
The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Add Subtag

The Add Subtag refactoring allows you to add a subtag to a set of XML tags. If this refactoring is invoked, subtags matching with requested subtag name will be added to tags with the selected tag name. Optionally, the requested subtag may be added only to tags that do not already contain the selected subtag. This bulk addition of subtags is useful as XML schemas evolve.

Adding a subtag

1. In the editor, place the cursor within the corresponding tag.
2. Select Refactor | XML Refactorings| Add Subtag from the main or the context menu.



3. Determine the name of the subtag to add to the selected tags.
4. Determine whether tags that already contain the requested subtag will be ignored, or will have their values changed.
5. Determine the scope of the addition. All tags matching name of the selected will be replaced, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
6. Press Preview button to make IntelliJ IDEA to search for usages of the selected subtag Find window.
7. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Pressing Preview opens the Refactoring preview window displaying all found usages of the subtags to be replaced. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

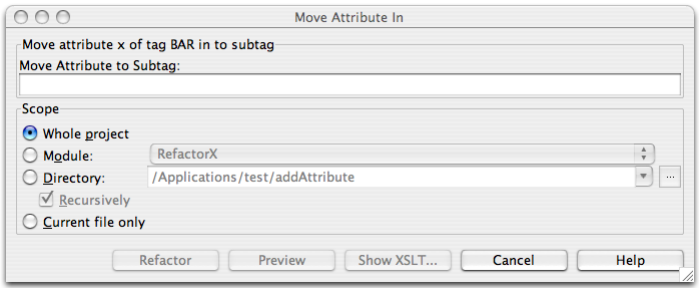
The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Move Attribute In

The Move Attribute In refactoring allows you to move attributes defined on a set of XML tags inward to a set of subtags. If this refactoring is invoked, all attributes matching the selected attribute name on tags with the selected tag name may be moved inward toward a subtag of a given name. This bulk modification of attribute values may be useful as XML schemas evolve.

Moving an attribute into a subtag

1. In the editor, place the cursor within the attribute to be moved.
2. Select Refactor | XML Refactorings | Move Attribute In from the main or the context menu.



3. Determine the subtag to move the attribute to.
4. Determine the scope of the change. All attributes matching the name of the attribute selected on the tag selected will be modified, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
5. Press Preview button to make IntelliJ IDEA to search for usages of the selected attribute Find window.
6. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Pressing Preview opens the Refactoring preview window displaying all found usages of the attributes to be changed. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

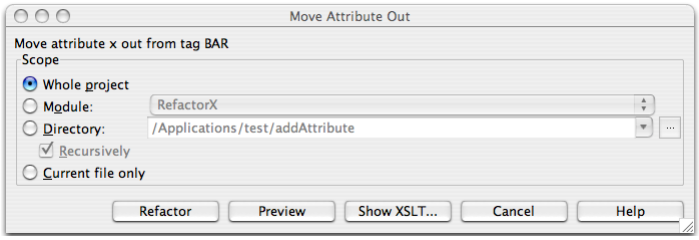
The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Move Attribute Out

The Move Attribute Out refactoring allows you to move attributes defined on a set of XML tags outward to their parent tags. If this refactoring is invoked, all attributes matching the selected attribute name on tags with the selected tag name may be moved outward. This bulk modification of attribute values may be useful as XML schemas evolve.

Moving an attribute to a parent tag

1. In the editor, place the cursor within the attribute to be moved.
2. Select Refactor | XML Refactorings | Move Attribute Out from the main or the context menu.



3. Determine the scope of the change. All attributes matching the name of the attribute selected on the tag selected will be modified, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
4. Press Preview button to make IntelliJ IDEA to search for usages of the selected attribute Find window.
5. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Pressing Preview opens the Refactoring preview window displaying all found usages of the attributes to be changed. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

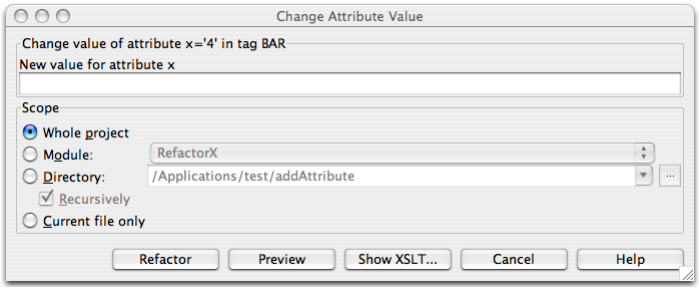
The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Change Attribute Value

The Change Attribute Value refactoring allows you to change the values an attribute defined on a set of XML tags. If this refactoring is invoked, all attributes matching the selected attribute name and selected attribute value on tags with the selected tag name may have their values changed. This bulk modification of attribute values may be useful as XML schemas evolve.

Changing a value of an attribute

1. In the editor, place the cursor within the attribute whose value you want to change.
2. Select Refactor | XML Refactorings | Change Attribute Value from the main or the context menu.



3. Determine the new value for the attribute.
4. Determine the scope of the change. All attributes matching the name and value of the attribute selected on the tag selected will be modified, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
5. Press Preview button to make IntelliJ IDEA to search for usages of the selected attribute Find window.
6. Cstepck OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Pressing Preview opens the Refactoring preview window displaying all found usages of the attributes to be changed. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

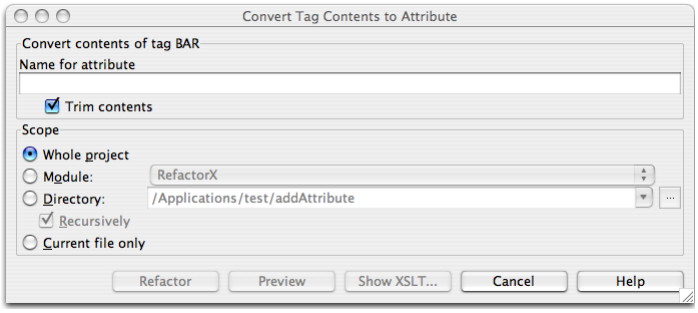
The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Convert Contents To Attribute

The Convert Tag Contents to Attribute refactoring allows you to replace the contents of a set of XMLs tag with equivalent attributes. If this refactoring is invoked, all tags matching the selected tag's name will have their textual contents removed, and equivalent attributes created. This bulk transformation of tag contents to attributes is useful as XML schemas evolve.

Converting tag contents into attributes

1. In the editor, place the cursor within the tag whose contents you want to convert.
2. Select Refactor | XML Refactorings | Convert Tag Contents to Attribute from the main or the context menu.



3. Determine the name of the attributes to be created from the selected tag contents.
4. Determine the scope of the replacement. All tags matching name of the selected will have their contents converted to attributes, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
5. Press Preview button to make IntelliJ IDEA to search for usages of the selected tag Find window.
6. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Pressing Preview opens the Refactoring preview window displaying all found usages of the tags to be converted. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Expand Tag

The Expand Tag intention allows you expand an empty XML tag into an equivalent start tag and end tag.

Collapse Tag

The Collapse Tag intention allows you collapse an XML tag whose contents are only white space, into an empty XML tag.

Merge Tags

The Merge Tag intention allows you to merge two sequential and compatible XML tags into one larger tag. The tags to merge must have the same name and attributes. To merge two tags, simply select the first tag to merge, and invoke the intention with `Alt+Enter`. Then select Merge Tags, and the tags will be merged. Merging effectively replaces the two tags with a new tag. The new tag will have the same attributes as the original tags, and its contents will be created by appending the contents of the first tag with the contents of the second.

Split Tags

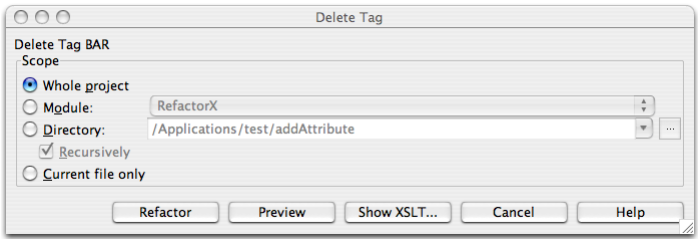
The Split Tag intention allows you split a large XML tag into two smaller XML tags. To split tags, simply select the child tag you wish the split to occur on in the body of the tag, and invoke the intention with `Alt+Enter`. Then select Split Tags, and the tag will be split. Splitting a tag will create two sequential tags, both with the same name and attributes as the original tag. The contents of the first tag will be whatever comes before the selected child tag, with everything else becoming the contents of the second tag.

Delete Tag

The Delete Tag refactoring allows you to delete a set of XML tags. If this refactoring is invoked, all tags matching the selected tag name on tags with the selected tag name may be removed. This bulk removal of tags may be useful as XML schemas evolve.

Deleting a tag

1. In the editor, place the cursor within the tag to be deleted.
2. Select Refactor | XML Refactorings | Delete Tag from the main or the context menu.



3. Determine the scope of deletion. All tags matching name of the tag selected will be deleted, if they are in scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
4. Click Preview to make IntelliJ IDEA search for usages of the selected tag.
5. Click OK to continue. If you do not select the Preview check box, all usages will be changed immediately.

Clicking Preview opens the Refactoring preview window displaying all found usages of the tags to be removed. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

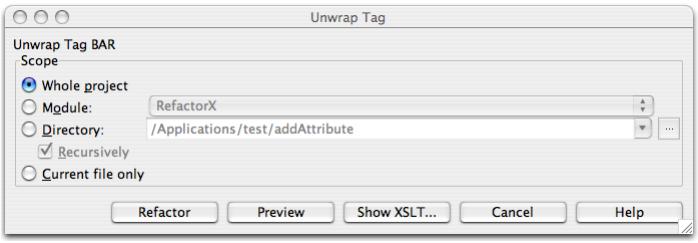
The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Unwrap Tag

The Unwrap Tag refactoring allows you to unwrap a set of XML tags, replacing them with their contents, if any. If this refactoring is invoked, all tags matching the selected tag name may be unwrapped. This bulk unwrapping of tags may be useful as XML schemas evolve. Note that top-level tags will not be unwrapped, as this may make XML documents invalid.

Unwrapping a tag

1. In the editor, place the cursor within the tag to be unwrapped.
2. Select Refactor | XML Refactorings | Unwrap Tag from the main or the context menu.



3. Determine the scope of the unwrapping. All tags matching name of the tag selected will be unwrapped, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
4. Press Preview button to make IntelliJ IDEA to search for usages of the selected tag Find window.
5. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Please note the following:

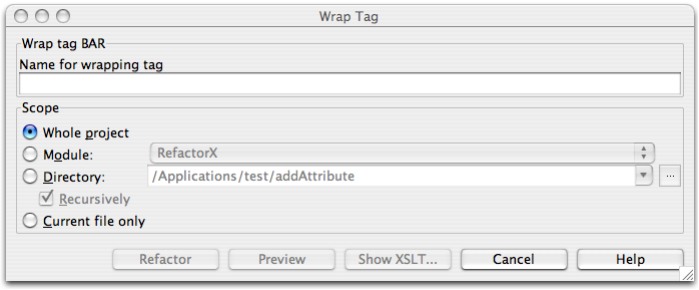
- Pressing Preview opens the Refactoring preview window displaying all found usages of the tags to be unwrapped. In this window you can exclude/include usages you want to refactor.
- Pressing Show XSLT... opens the XSLT preview window displaying a small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.
- The Refactoring preview window may appear anyway, if the files to be affected are read-only.

Wrap Tag

The Wrap Tag refactoring allows you to wrap a set of XML tags in a newly created parent. If this refactoring is invoked, all tags matching the selected tag name may be wrapped. This bulk wrapping of tags may be useful as XML schemas evolve.

Wrapping a tag

1. In the editor, place the cursor within the tag to be wrapped.
2. Select Refactor | XML Refactorings | Wrap Tag from the main or the context menu.



3. Determine the name of the new tags which will wrap the selected tags.
4. Determine the scope of the wrapping. All tags matching the name of the tag selected will be wrapped, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
5. Press Preview button to make IntelliJ IDEA to search for usages of the selected tag Find window.
6. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Please note the following:

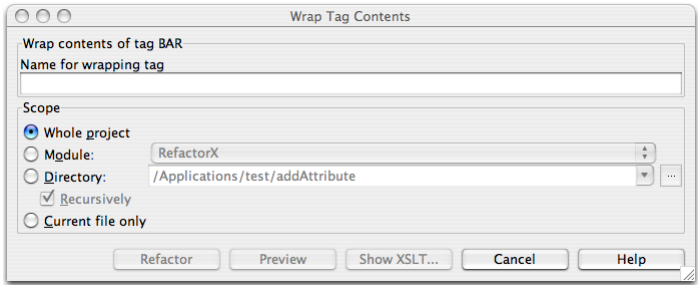
- Pressing Preview opens the Refactoring preview window displaying all found usages of the tags to be wrapped. In this window you can exclude/include usages you want to refactor.
- Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.
- Refactoring preview window may appear anyway, if the files to be affected are read-only.

Wrap Tag Contents

The Wrap Tag Contents refactoring allows you to wrap the contents of a set of XML tags in newly created tag. If this refactoring is invoked, all tags matching the selected tag name will have their contents wrapped. This bulk wrapping of tag contents may be useful as XML schemas evolve.

Wrapping tag contents

1. In the editor, place the cursor within the tag whose contents you want to wrap.
2. Select Refactor | XML Refactorings | Wrap Tag Contents from the main or the context menu.



3. Determine the name of the new tags which will wrap the contents of the selected tags.
4. Determine the scope of the wrapping. All tags matching name of the tag selected will be wrapped, if they are in the selected scope. Scopes available include the current file, the entire project, or a specified directory or module. Directory scopes can either include sub-directories, or not, based on whether the Recursively check box is selected. If the Limit to files with DTD:... check box is selected, the scope will be limited to those files with the same DOCTYPE as the current file.
5. Press Preview button to make IntelliJ IDEA to search for usages of the selected tag Find window.
6. Click OK to continue. If you do not select the Preview option, all usages will be changed immediately.

Pressing Preview opens the Refactoring preview window displaying all found usages of the tags to be wrapped. In this window you can exclude/include usages you want to refactor.

Pressing Show XSLT... opens the XSLT preview window displaying an small fragment of XSLT equivalent to the refactoring requested. This fragment can be used by XSLT processors to perform the requested refactoring on files external to your IntelliJ IDEA project.

Refactoring preview window may appear anyway, if the files to be affected are read-only.

XPath and XSLT Support

In this section:

- XPath and XSLT Support
  - Prerequisites
  - Overview
  - Changes to the UI
- XPath Expression Evaluation
- XPath Search
- XPath Expression Generation
- Plugin Settings
- XSLT Support

Prerequisites

Before you start working with XSLT, make sure that the XPath View+XSLT Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

- XSLT support is available in all XML files that declare the XSLT-Namespace `http://www.w3.org/1999/XSL/Transform` on their root element.
- IntelliJ IDEA supports XSLT versions 1.0 and 2.0, XPath versions 1.0 and 2.0.
- An XSLT 2.0-capable processor (e.g. Saxon 9-HE) should be added to the classpath of a module that is used to run a stylesheet, and placed in front of JDK. See Order of dependencies.

Overview

The plugin lets you do the following:

- Evaluate expressions against the currently focused document, including support for the `document()` function to make cross-document queries
- Evaluate an expression against multiple XML documents in a *Find in Path* style way by the new Find by XPath action.

The Expression Evaluation dialog allows you to highlight matching expressions in the current editor or to open the Find tool window to display a list of matching lines. Editing XPath expressions is enhanced by on-the-fly error-checking including a set of customizable XPath Inspections and a wide range of code completion suggestions.

The plugin also offers a way to display a unique XPath expression for a selected element in an editor. This is available via View | Unique XPath in the Main Menu.

XSLT support is not limited to XPath expressions, it also supports a wide range of XSLT constructs, like checking the existence of templates that are called via `xsl:call-template` and their parameters, refactoring and navigation enhancements. Find out more about this in the XSLT Support section.

Changes to the UI

- XPath viewer settings
- XSLT settings
- XSLT run/debug configuration

XPath Expression Evaluation

Evaluating XPath expressions in IntelliJ IDEA has two main purposes: Testing XPath expressions that are to be used in program code or XSLT scripts, and making structured queries against XML documents. The Evaluate XPath action can be invoked either from the editor context menu, the main Search Menu or the main toolbar by clicking the action's icon .

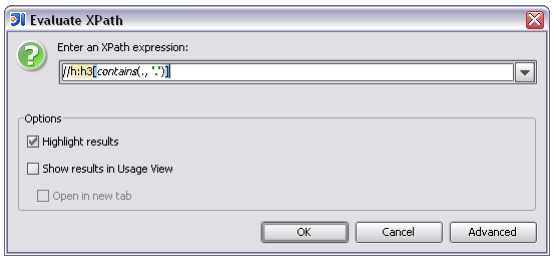
The Expression Evaluation has two different modes, a *simple* mode, that allows you to enter simple one-line expressions and doesn't allow you to configure and Context settings. For more convenient editing of long expressions in a multi-line style way, there's the *advanced* mode dialog that also has a button to edit some context settings, such as namespaces and their prefixes, and variables to use for the evaluation.

Both dialogs feature a history of the recently evaluated expressions, completion, syntax-checking and highlighting, as well as some semantic error checking of the entered expression. Semantic checks include validation of used namespace prefixes, useless XPath expressions (e.g. `@comment()`) and node tests for element/attribute names that don't occur in the context document and would not be successfully matched.

Some error checks and XPath inspections also provide Quick Fixes for detected problems, e.g. the possibility to map an unresolved namespace-prefix to a URI by intention.

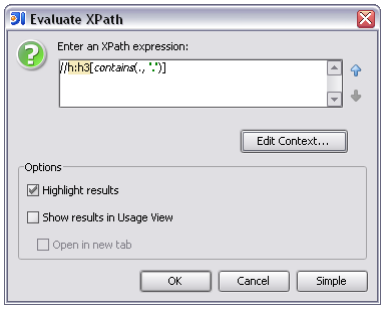
Simple Mode

The simple mode comes with an input field that can be used to enter simple one-line expressions that don't require any customization of namespace prefixes or make use of predefined variables. The last recently used expressions can be selected from the drop-down list.



Advanced Mode

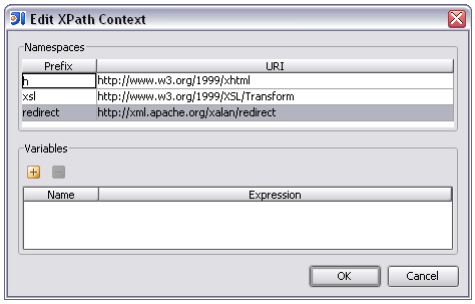
The advanced mode adds the possibility to edit expressions in a multiline editor and has another button to edit the XPath Context . The expression history can be browsed by the Up- and Down arrays on the right of the dialog or the keyboard shortcuts for the previous/next history element (usually Ctrl-Alt-Up, Ctrl-Alt-Down).



XPath Context

This dialog allows you to assign custom prefixes to the namespace URIs that are used in the context document. This can be useful to assign a shorter prefix, resolve prefix clashes or to actually define a prefix for the default namespace. This can be essential because XPath does not automatically match elements in the default namespace without specifying a prefix for the element to be matched.

The XPath Context also includes the possibility to define custom XPath variables that can be used in queries for repeating expressions. Each variable in the table can be assigned an expression that will be evaluated once when the query is executed. The resulting value is then available for multiple use at no additional computational cost.



Options

Highlight results

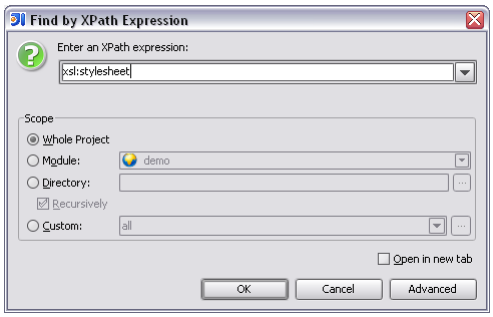
This highlights the matched nodes in the current editor. Matched nodes that don't belong to the current editor (may happen by using the `document()` function) are not highlighted. It's recommended to display such cross-document results in the Find Usages toolwindow.

Show results in Usage View

Shows all matched nodes in the Find Usages toolwindow. Check Open in new tab to open the result in a new tab instead of reusing the last one.

XPath Search

This new feature is the XML-aware counterpart to IntelliJ IDEA built-in functions *Find in Path* and *Search Structurally*: It allows you to find occurrences of certain XPath expressions in all XML files in a specific scope. It is available as Find by XPath... under the Search menu.



The scope that can be chosen is either the full project scope, a specific module or a simple directory. There's also the ability to use *custom* scopes, but this is somewhat limited as it will only scan XML files inside source-folders.

This dialog provides the same functionality regarding completion, error checking and intentions as described in Evaluate Expression .

XPath Expression Generation

This action computes a unique XPath expression that matches the currently selected node in the document. The action is available from the View-Menu (Unique Path) and the Editor-Context Menu (Show Unique XPath). The action is only enabled when the caret is placed on an element that a useful expression can be generated for.

Tip The generated expression in the pop-up can be selected and copied into the clipboard for further use.

If a simple XPath expression like `/root/something/else` doesn't produce a unique result, the action has two strategies to make it unique:

- If the non-unique node is an element, the action looks for attributes with the name 'id', 'name' and attributes that are of ID-type, as defined by the document's DTD or XML Schema.
  Example: `/root/something[@id="foo"]/else`
- For nodes other than elements (comments, processing instructions), or if the above rule doesn't produce a unique result, the index of the node inside its parent is appended.
  Example: `/root/something/else[2]`

Plugin Settings

XPathView plugin settings are configured on the XPath Viewer page of the Settings .

Item	Description
Scroll first hit into visible area	Select this check box to have the editor automatically scroll to the first XPath match.
Use node at cursor as context node	Select this check box to have the entered XPath expression use the currently selected node (tag/attribute/pi, etc.) as its context node and evaluate the expression relatively to this node.
Highlight only start tag instead of whole tag content	Do one of the following: − Select this check box to have only the name of a matching tag highlighted. − Clear this check box to have the entire content of a matching tag highlighted.
Add error stripe markers for each result	Select this check box to have each match supplied with an error stripe marker which can be quickly navigated to. The tooltip of each marker shows the matched content.
Show actions in Toolbar	Select this check box to have buttons that invoke XPath-related actions displayed on the Main Toolbar .
Show actions in Main Menu	When this check box is selected, XPath-related actions are available from the main menu.
Colors	In this area, configure color indication during execution of XPath expressions. Clicking on the color box will open the Select Color dialog in which you can modify the current color indication. − Highlight Color - in this area, select the color to indicate XPath matches in the editor. − Context Node Color - in this area, select the color to indicate the current context node.

XSLT Support

- Completion
- Refactoring
- Error Highlighting
- Navigation
- Documentation
- Run Configurations
- File Associations
- Intentions

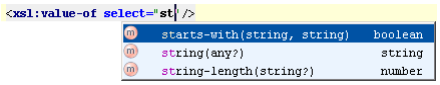
Completion

With Code Completion being one of the key features of IntelliJ IDEA, the plugin provides several possibilities to complete keywords, predefined functions, variables and parameters used in XPath expressions, template names and names of parameters that can be passed to a template invocation.

Completion in XPath expressions

It's possible to complete all parameters/variables in scope inside an XPath expression in a normal expression attribute or inside an attribute value template

Also, all predefined functions and keywords are available for completion, including function signatures.



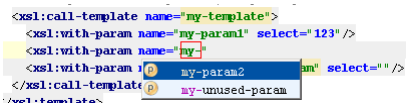
The Quick Documentation Lookup also works in completion lookup lists.

Completion for template names in xsl:call-template

The template's name that is to be called can be completed from a list of all named templates in the current document and included stylesheets

Completion for template parameters

There's a special completion for parameters that should be passed to a template in a `xsl:call-template` invocation. The completion lists all parameters that are declared by the template and are not yet present in the *argument list* of the invocation, i.e. there's no `xsl:with-param` yet.



Tip That's especially useful when also using completion to create the `xsl:with-param` tag, because this will automatically trigger the completion for the parameter's name which is required according to the schema; e.g.

`<xsl:with-p` <ctrl-space>

=>

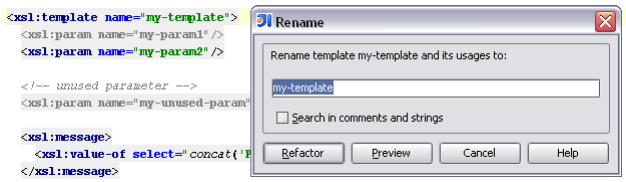
`<xsl:with-param name="` [lookup list] `"`

Refactoring

The plugin supports some of IntelliJ IDEA built-in refactorings Rename and Safe Delete for XSLT items such as templates, variables and parameters. It's also possible to create XSLT variables from selected XPath expressions with the well-known Introduce Variable shortcut. Inlining variables is possible with the Inline shortcut as well.

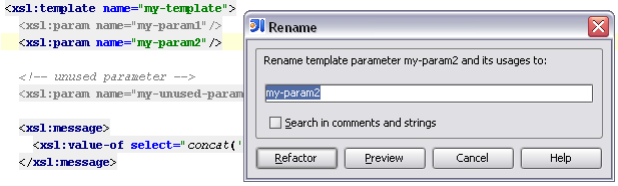
Renaming Templates

Named templates can be renamed in IntelliJ IDEA just like any other symbol. All `xsl:call-template` invocations that refer to this template will be updated accordingly.



Rename Variables and Parameters

Just as named templates, it is possible to rename XSLT variables and template parameters, either at a point of their use or at their declaration.



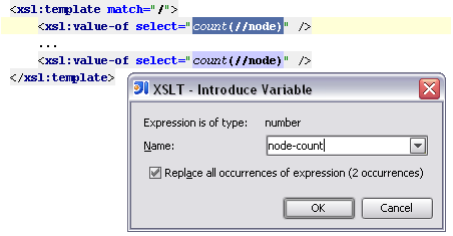
Safe Delete

Named templates, parameters and variables can be deleted using IntelliJ IDEA Safe Delete feature, i.e. the item will be removed if there aren't any references left to it in other stylesheets across the project.

This is especially useful if the stylesheet may be included in other ones via `xsl:include` or `xsl:import` to make sure that nothing will be deleted that is still used somewhere else.

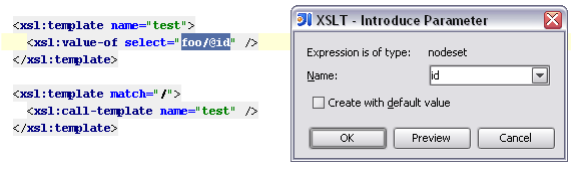
Introduce Variable

It is possible to extract XPath-Expressions and turn them into an xsl:variable declaration. Check the Replace all occurrences check box to replace all other occurrences of the same expression.



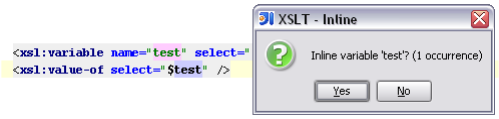
Introduce Parameter

This is similar to Introduce Variable, but it creates a new parameter instead of a variable. It also has an additional option Create with default value that determines if the selected expression should be added as the introduced parameter's default value or if all calls to the template should be updated to pass the selected expression. That option is only available when introducing parameters to named templates.



Inline Variable

This is just the opposite of Introduce Variable, it replaces all usages of a variable with the expression that is specified in the variable's select-attribute. Variables that don't have such an attribute cannot be inlined. Inlining variable references that resolve to parameters is not possible as well.



Error Highlighting

The XSLT support can detect a range of errors in XSLT constructs, such as misspelled template names, missing template parameters, bad match-patterns, references to undeclared variables, wrong or useless embedded XPath expressions, etc. and also offers Quick Fixes to automatically fix some of those errors.

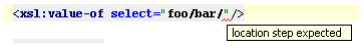
– Syntax Highlighting
– XPath Syntax checks
– XPath Type Checking
– Pattern validation
– Unresolved References
– Duplicate declarations
– Other checks

Syntax Highlighting

Where allowed, XPath function-calls, Axis names, Numbers, Strings, etc. are highlighted according to the currently active color scheme. By default, the plugin uses the colors that are defined for the corresponding Java types, such as number- and string literals, etc. If a different coloring is desired, those colors can be configured in IDE-Settings | Colors & Fonts on the tab XPath.

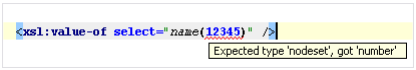
XPath Syntax checks

Just like the interactive XPath Expression Evaluation, the XSLT support catches any syntax error in XPath expressions used inside a stylesheet.



XPath Type Checking

In XPath, almost all types are assignable to each other with certain well-defined conversion semantics. However, the conversion to NODESET isn't defined for any type and there's also no (portable) conversion function available. Such type-conversions are highlighted as an error.



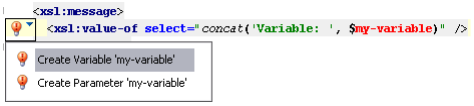
Pattern validation

A special form of XPath expression are patterns in XSLT. They are e.g. used as the value of the *match*-attribute in `xsl:template` elements. Only a certain subset of XPath expressions is allowed here, which the XSLT support checks for.

Unresolved References

References to variables that have not been declared or are not accessible from the current scope are detected and highlighted as an error. There are Quick Fixes available to create a variable or parameter declaration for such unresolved variables references.

Quick-Fixes:



Duplicate declarations

In XSLT there must not be more than one variable or parameter declared on the same scope level. It's also not allowed that there is more than one template with the same name. The plugin will identify such duplicate declarations and highlight them in the editor.

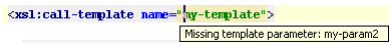
Other checks

Shadowed variables

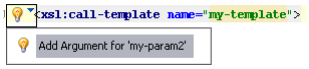
Even though is possible to have identically named variables or parameters in different nesting levels, this can be confusing and is likely to cause programming mistakes. The plugin can identify shadowed declarations and offers Quick-Fixes to either rename the local or the outer variable.

Missing Template Arguments

Another check that the XSLT support performs is whether all required parameters are specified with a `xsl:call-template`. A parameter is considered required if there is no default value, i.e. if there's no *select* attribute and the paremeter's declaring element has an empty body.



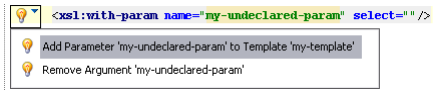
Quick-Fixes:



Superfluous Template Arguments

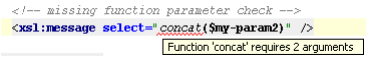
There's also a supplemental check that flags arguments that are not declared as template parameters. There are Quick Fixes available to either remove the argument from the template-call or to add a corresponding parameter to the called template.

Quick-Fixes:



Function Call Arguments

The XSLT support does, just like the interactive XPath Expression Evaluation, check whether the number and types of function arguments match their declaration for built-in functions of XPath and XSLT.



XPath Inspections

All XPath Inspections are supported for editing XSLT documents. Those inspections can also be suppressed in a way that is similar to the standard suppression-mechanism IntelliJ IDEA uses for Java code by using *noinspection* XML comments. The suppression is possible on different levels, either on instruction level, template-level (if applicable) or stylesheet level.

XPath Inspections

The XPath language implementation provides a few built-in inspections that can check for common coding mistakes when writing XPath expressions both in the interactive mode and when writing XSLT scripts. Those inspection also provide a number of configuration options which can be configured on the Inspections page of the Settings dialog box.

Due to the way how these inspections are implemented and integrated, they only work for the on-the-fly editor highlighting and *not* if the inspections are run via Analyse | Inspect Code .

XPath Type Checking

There are two inspections that deal with type-conversion in XPath expressions: Implicit Type Conversion and Redundant Type Conversion.

Implicit Type Conversion

This inspection checks for any implicit conversions between the predefined XPath-types STRING, NUMBER, BOOLEAN and NODESET. While this is usually not a problem as the conversions are well-defined by the standard, this inspection can help to write XSLT scripts that are more expressive about types and can even help to avoid subtle bugs:

`<xsl:if test="foo" />` is not the same as `<xsl:if test="string(foo)" />`

The first test checks whether the element *foo* exists (`count(foo) > 0`), the latter one however is only true if the element actually contains any text (`string-length(foo) > 0`). The plugin will then offer to make the type-conversion more explicit.

There are several options to adjust the inspection to personal preferences by offering the possibility to individually enable it for implicit conversions between certain types.

The plugin can also be told to always flag explicit conversions that do not result in the actually expected type, such as `<xsl:if test="number(foo)" />` and provides a special option to ignore the conversion from NODESET to BOOLEAN by using the `string()` function as a shortcut for writing `string-length() > 0` .

Redundant Type Conversion

This inspection checks whether any type-conversion with the functions `string()` , `number()` or `boolean()` is redundant, i.e. whether the type of argument is the same as the functions return type or if the expected type of the expression is of type *any*. While such an explicit conversion may sometimes be intentional to emphasize the type, this can usually be safely removed.

Expression validity checks

Those inspections check whether an expressions contains any potential semantic mistakes, such as referencing element/attribute names that don't occur in instance documents or using predicates that don't potentially match anything.

Check Node Test

This inspection checks whether any element/attribute names that are used in XPath-expressions are actually part of an associated XML file or are defined in a referenced schema. This helps to avoid problems caused by typos in XPath-expressions that would otherwise occur when running the script and may even then not be recognized immediately.

Example:

```
<xsl:template match="h:txtarea" />
```

If the prefix *h* is bound to the XHTML namespace, the inspection will flag this part of the match-expression as an unknown element name because the correct name of the element is *textarea*.

Index Zero Usage

This inspection checks for any accidental use of zero in a predicate index or in a comparison with the function `position()` . Such is almost always a bug because in XPath, the index starts at one, *not* at zero.

Example:

`//someelement[position() = 0]` or `//something[0]`

Developing custom XPath inspections

The XPath inspections make use of the normal inspections API of IntelliJ IDEA. However, due to the way the XPath Language-Support is integrated, this is a bit more complicated and it's at the moment not readily possible to develop full-blown 3rd-party XPath inspections. While it's theoretically possible to develop custom inspections that make use of the XPath-PSI API and are derived from `org.intellij.lang.xpath.validation.inspections.XPathInspection` , this is not recommended and not supported.

Please contact me if there's a need for a special inspection or there is significant interest that would justify the effort to make this more pluggable.

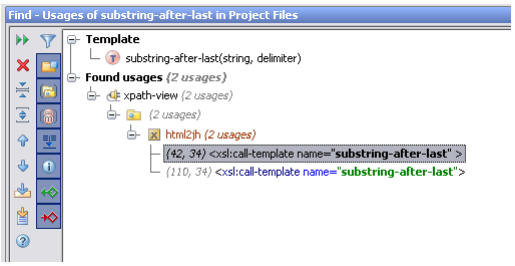
Navigation

Basically all possibilities for code-navigation are supported for named templates, variables and parameters. For example: Goto Declaration, Find Usages, Highlight Usages in File, Quick Definition Lookup, etc.

```
<xsl:variable name="var" select="123" />
<xsl:value-of select="$var + $var" />
```

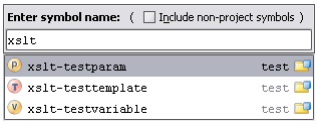
Find Usages

Find Usages works for Variables and Parameters as well as named XSLT Templates. It finds all places of a certain template in all XSLT Stylesheets that are contained in the specified scope.



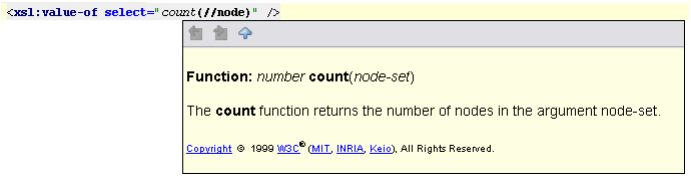
Goto Symbol

The plugin also supports IntelliJ IDEA Goto Symbol action. The names of templates, top-level variables and parameters are offered for that action to be able to quickly navigate to them by their name.



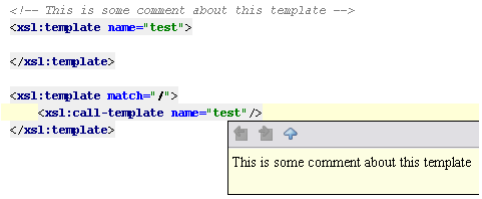
Documentation for XSLT-Elements and predefined XPath-Functions

The plugin comes with bundled documentation for XSLT elements and functions that is accessible via the Quick Documentation Lookup (Ctrl-Q) action. This documentation is extracted from the respective sections in the official W3C XSLT and XPath documents.



Custom Documentation

It's also possible to annotate elements in an XSLT stylesheet with some documentation fragments. This can be looked up by the Quick Documentation Lookup function for variables, parameters and templates



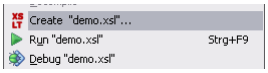
Run Configurations

Running XSLT Scripts is as easy as opening the Editor Context Menu and either creating a permanent Run Configuration or simply choosing Run to instantly run the selected XSLT script.

Creating Run Configurations

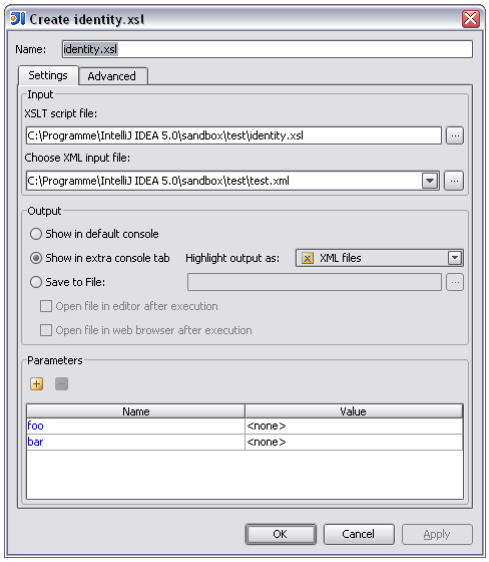
Permanent Run Configurations can be created through Main Menu | Run | Edit Configurations. You'll find an additional tab named XSLT to add Run Configurations for XSLT scripts.

A Run Configuration can also be created by the Create "<name>" action from the context menu.



In that case, the name of the configuration will be taken from the stylesheet's file name and its path will already be filled into the XSLT script file text field. If the stylesheet defines parameters, those will be filled into the Parameters table with empty values. Parameters that are highlighted in blue don't have a value assigned yet and will not be passed to the stylesheet during execution, so it's not required to delete those automatically created parameter values if nothing should be passed to the stylesheet.

Run Configuration Settings



Input

An XSLT Run Configuration has various settings that can be adjusted. The most important ones are the location of the XSLT script file and the XML input file that should be transformed. Those are mandatory and the specified files must exist, otherwise the configuration cannot be executed.

The XML input file combobox lists all XML files that have been associated with the chosen stylesheet via the File Associations functionality.

Output

There are three different choices about how the output of the script should be handled. The first is Show in default console. When this is selected, the output will appear in the normal run console, together with any warnings and error messages from the XSLT transformer, as well as messages generated by the script, e.g. by `xsl:message`.

By default selected is the option Show in extra console tab which will show the output in an extra tab named *XSLT Output*. This option has the ability to highlight the produced output according to different file types that are available in IntelliJ IDEA. However this is kind of an experimental feature, so it can be turned off completely by selecting the Disabled option. The output will then be written into a temp file that is displayed by the normal Log Viewer.

The last option, Save to file can be used to directly write the output to a file. The textfield must not be empty and can refer to any existing or non-existing file. Check the option Open file in editor after execution to open the file in IntelliJ IDEA after the script ends normally. The option Open file in web browser can be checked to open the generated file in the configured web browser after execution has finished.

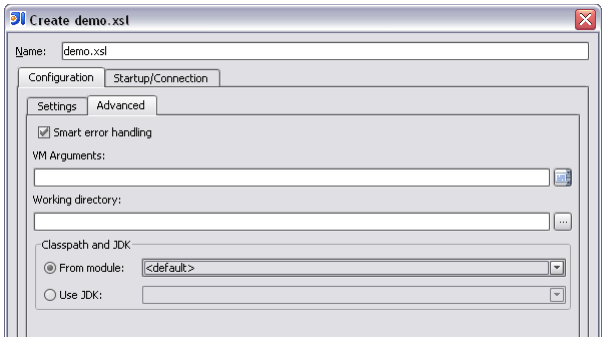
Warning! The specified output file will be overwritten without confirmation.

Parameters

The Parameters table is used to specify the parameter names and their values that should be passed to the script. Press the Add and Remove buttons to modify the list. A newly added parameter will not have any value assigned by default, and thus will not be passed to the script if the value isn't edited.

Advanced Options

This tab allows you to control some options that are not needed for usual run configurations.



Smart error handling

Uncheck this to see full error messages including their complete stack traces when an error occurs during execution. When checked, those stack traces will be suppressed and only the relevant information about errors will be displayed in the console.

VM Arguments

Allows you to pass arbitrary VM arguments to the VM that is used for running the XSLT script.

Working Directory

The working directory to use. When left empty, the working directory will be the directory the XSLT script file is located in.

Classpath and JDK

Allows you to choose the environment to run the script under. The default setting is the Module the XSLT script file belongs to. The option From module will also include the full classpath of the chosen module. This can be needed if the script makes use of custom XSLT Extension Functions.

The option Use JDK allows to select the JDK without including anything module- or project-related into the classpath. It can be useful to explicitly choose a specific JDK to test the script with.

File Associations

File Associations are used to associate an XSLT file with other XML files. This is currently used for three purposes:

- Enhanced completion for element- and attribute names in XSLT node-selections. The completion will offer all element- and tag names that are found in the associated documents.
- Enhanced error highlighting for XSLT node-selections. If an XSLT script has been associated with one or more XML files, any references to element- and attribute names that are not part of the associated files fill be flagged with the warning message: " Tag name '...' is not part of the document".
- File Associations are also used for creating Run Configurations. The XML input file to use for the transformation can be conveniently chosen from the list of associated files.

Managing File Associations

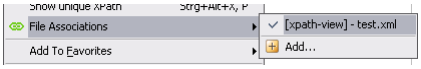
XSLT File Associations are defined per project and managed in the XSLT File Associations settings page.

Managing Associations from the editor

Associations can be also created by invoking the Add... action from the File Associations group in the Editor Context Menu. A file-selection dialog opens that can be used to select one or more X(HT)ML files to associate the XSLT file with.

Removing an Association

An Association can be removed by clicking on the corresponding file name in the File Associations group. The file names are displayed with a path that is relative from the current file. If the associated file is part of any module, the module name is included in square brackets.



It's also possible to invoke the associations configuration dialog through the Configure... action. This opens the associations configuration dialog and preselects the file that's currently opened in the editor.

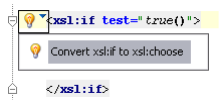
Intentions

The XSLT Support comes with a few intentions that can be useful for XSLT development.

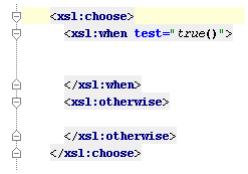
Intention to convert xsl:if to xsl:choose

This is useful if it turns out that a simple if-branch isn't enough in a certain situation and an else-branch is required. When positioning the caret on the start tag of the `xsl:if`, the intention shows up and converts the code fragment into an `xsl:choose`, preserving the original `xsl:if` as an `xsl:when` block and adding an `xsl:otherwise` block.

Before



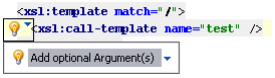
After



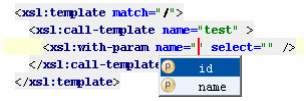
Intention to add optional parameters

This intention can be used to pass a value for an optional parameter in a template-call. It will insert the appropriate `xsl:with-param` tag and show a lookup list for all parameters that aren't already passed to the template.

Before



After



Tutorial: Using Emacs as an External Editor

On this page:

- What this tutorial is about
- Prerequisites
- Configuring Emacs as an external tool
- Opening current file in Emacs
- Assigning a keyboard shortcut

What this tutorial is about

This short tutorial aims to walk you step by step through defining Emacs as an external editor for IntelliJ IDEA.

Basics of Emacs are out of scope of this tutorial.

Prerequisites

Make sure that:

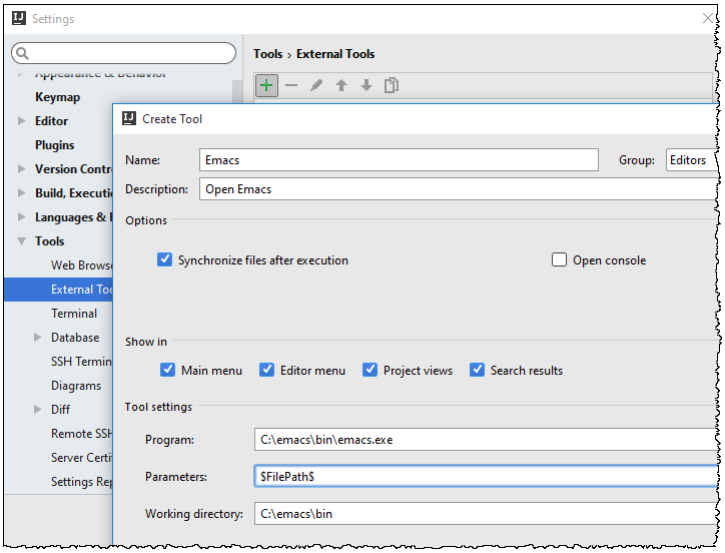
- You are working with IntelliJ IDEA 10.0 or later. This tutorial is created with IntelliJ IDEA version 2016.1.
- Emacs is downloaded and properly installed on your computer.

Configuring Emacs as an external tool

Open Settings/Preferences dialog. To do that, you can, for example, choose File | Settings (on Windows and *nix) or IntelliJ IDEA| Preferences (on Mac OS), or click button on the main toolbar.

Then, under the Tools node, open the page External Tools . On this page, you have to specify your Emacs installation as an external editor for the current file. This is how it's done...

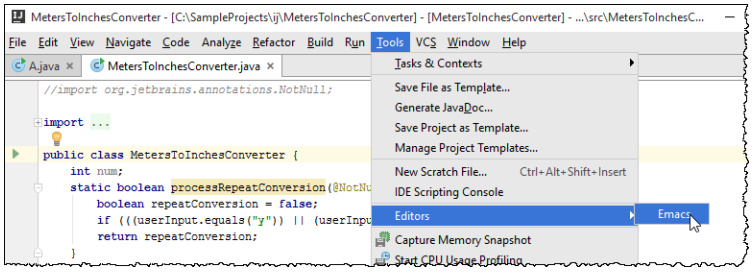
1. First, in the External Tools page, click +. The Create/Edit Tool dialog box opens.
2. In this dialog, do the following:
   - Type the tool name (Emacs) and optional description (Open Emacs)
   - Specify the name of the group, under which Emacs will appear in the Tools menu. In this example, the group name is Editors. This step is optional - if you specify no group name, then Emacs will appear in the Tools menu as is.
   - Clear the check box Open console.
   - Specify Emacs binary file location. You can either type it manually, or click the ellipsis button and find the desired binary in your file system.
   - Since you want to open the current file in Emacs, pass the file path as a parameter to the program: in the Parameters field, type `$FilePath$` .
   - Finally, specify the working directory - in our example, this is `$ProjectFileDir$`
   - Click OK.



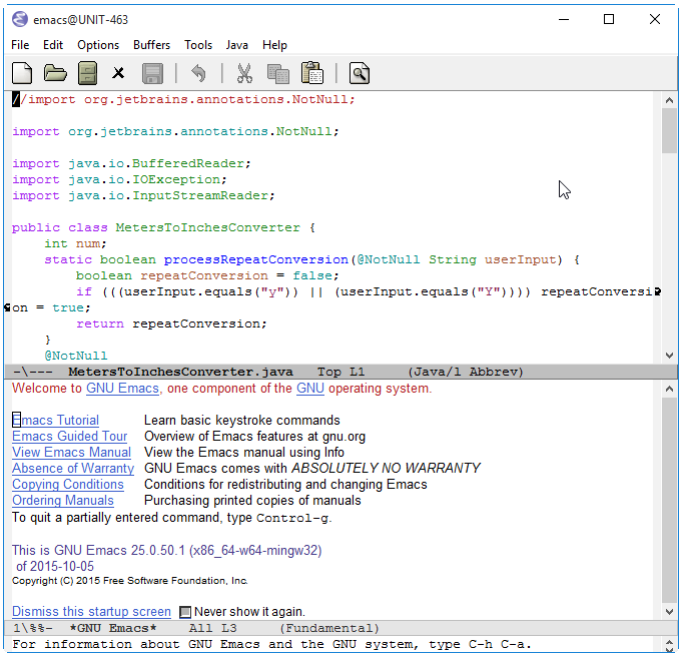
3. Apply changes and close the Settings/Preferences dialog.

Opening current file in Emacs

When you now look at the Tools menu, you will see the new node Editors. Pointing to this node reveals the Emacs command:

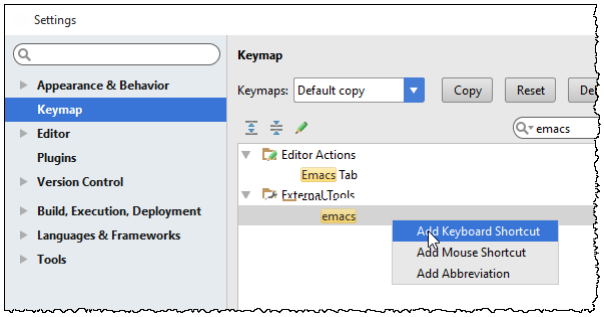


Open a file for editing . Next, on the Tools menu, choose Editors|Emacs - and see the current file in Emacs also:

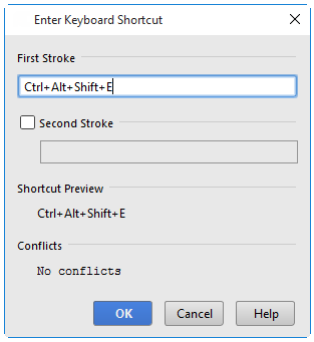


Assigning a keyboard shortcut

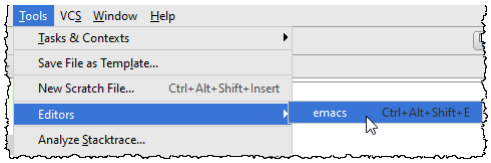
By the way, IntelliJ IDEA makes it possible to assign a keyboard shortcut to this action: click to open Settings/Preferences dialog, open the Keymap page , find Emacs, and choose Add Keyboard Shortcut on the context menu:



Enter Keyboard Shortcut Dialog dialog box opens, where you have to specify, which shortcut you would like this action to be associated with. Let's, for example, use `Ctrl+Shift+Alt+E`:



No conflicts are reported, so click OK, and see the new shortcut appearing in the list of actions and on the Tools | Editors | Emacs menu:



Tutorial: Using IntelliJ IDEA as the Vim Editor

On this page:

- Before you start
- Downloading and installing IdeaVim plugin
- What happens to IntelliJ IDEA's UI after restart?
- Configuring shortcuts
- Editing modes

Before you start

Make sure that:

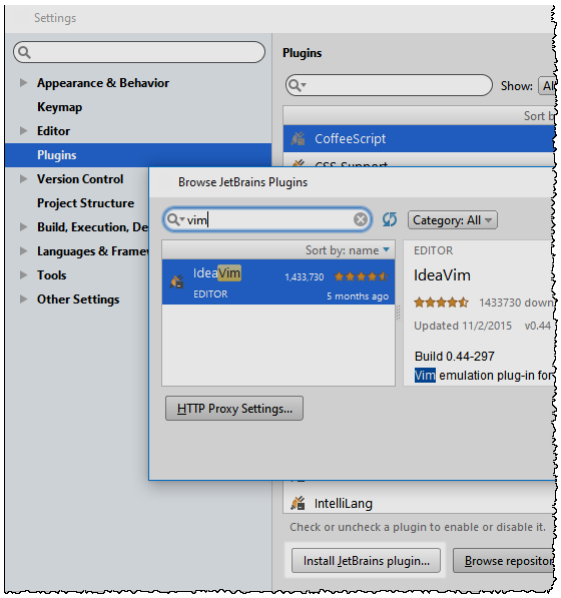
- You are working with IntelliJ IDEA version 15.0.0 or higher. If you still do not have IntelliJ IDEA, download it from this page . To install IntelliJ IDEA, follow the instructions, depending on your platform.

  This tutorial is created with IntelliJ IDEA version 2016.1.

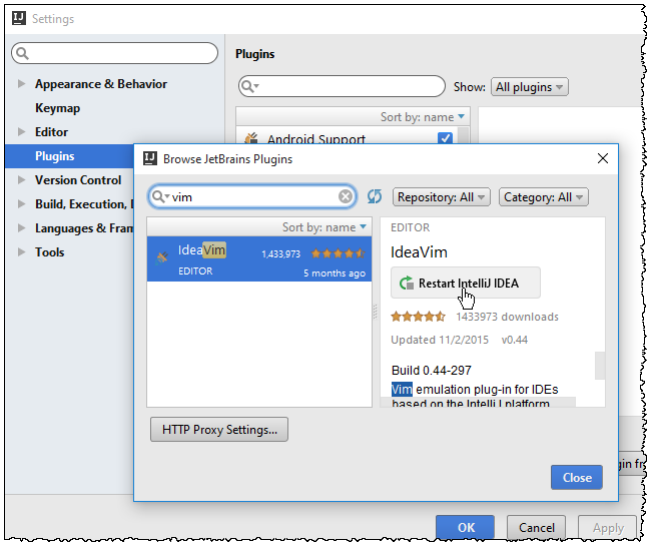
Downloading and installing IdeaVim plugin

On the toolbar of the IntelliJ IDEA main window, press `Ctrl+Alt+S` to open the Settings/Preferences dialog, and then click Plugins .

You see the list of plugins currently installed on you computer. However, the IdeaVim plugin is not among them. Click the button Browse JetBrains plugins. IntelliJ IDEA shows the contents of the huge JetBrains repository... you can type the word "vim" in the search field to narrow down the list:

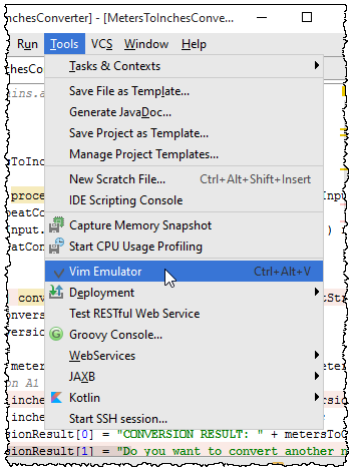


After installing the plugin, it actually becomes available after IntelliJ IDEA restart:



What happens to IntelliJ IDEA's UI after restart?

First, on the Tools menu, a check command Vim Emulator appears:



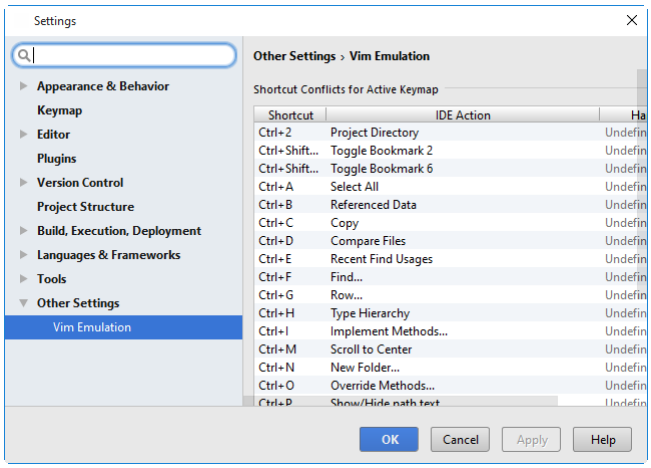
After IntelliJ IDEA restart, this check command is selected. You can disable Vim by clearing this check command.

Second, in Settings/Preferences dialog, an additional node Other Settings appears, with the page Vim Emulation. This page appears after restart!

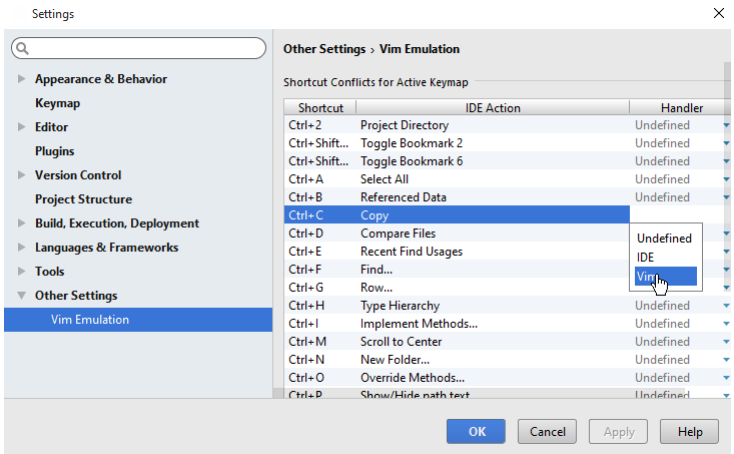
Configuring shortcuts

Both Vim and IntelliJ IDEA are keyboard-centric. With IdeaVim plugin, it is quite possible that IntelliJ IDEA's keymap runs into a conflict with the Vim keymap. That's why IntelliJ IDEA allows you choosing which keyboard shortcut you prefer for a certain action. This is how it's done.

Open Settings/Preferences dialog, and under the node Other Settings, click Vim Emulation:



In the Shortcut column, select the shortcut you want to configure. Next, in the Handler column, click the corresponding cell, and see the drop-down list of three possible options (Undefined, Vim, IDE):



If you choose IDE, it means that the IntelliJ IDEA's shortcut for this particular action is enabled. When you press, say, `Ctrl+Z`, IntelliJ IDEA silently performs its action.

If you leave the handler undefined, then, on pressing the shortcut, say, `Ctrl+B`, IntelliJ IDEA shows the following banner:

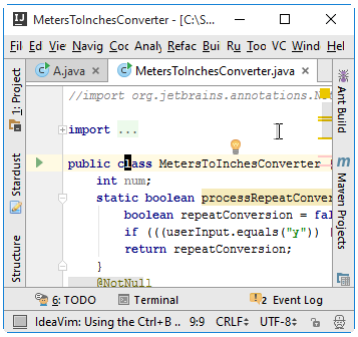
You can choose to redefine this shortcut as an IDE shortcut and thus accept the IntelliJ IDEA's keymap. To do so, click the link IDE shortcut.

If you click the link Vim Emulation, then IntelliJ IDEA will show the Vim Emulation page of the Settings/Preferences dialog.

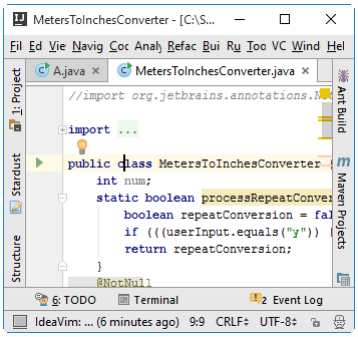
For the purposes of this tutorial, click the link Vim Emulation. Then, when you press `Ctrl+B`, IntelliJ IDEA will perform the Vim action for this keyboard shortcut.

Editing modes

OK, now that you have Vim enabled, you see that the cursor has changed its shape - now it is a block, which means that you are in the Normal mode :



If you want to enter the Insert mode , press `i` , and the cursor will turn into a line:



In this mode you can type new or change the existing code. Same way, you can enter the various Vim modes: for example, press `r` for the Replace mode .

By the way, as soon as you enter Vim emulation, it is also reported in the Status bar.

To return to the Normal mode, press `Escape` .

Migration Guides

- Eclipse
- NetBeans

Migrating From Eclipse to IntelliJ IDEA

Overview

Switching from Eclipse to IntelliJ IDEA, especially if you've been using Eclipse for a long time, requires understanding some fundamental differences between the two IDEs, including their user interfaces , compilation methods , shortcuts , project configuration and other aspects.

User Interface

No workspace

The first thing you'll notice when launching IntelliJ IDEA is that it has no workspace concept. This means that you can work with only one project at a time. While in Eclipse you normally have a set of projects that may depend on each other, in IntelliJ IDEA you have a single project that consists of a set of modules.

If you have several unrelated projects, you can open them in separate windows.

If you still want to have several unrelated projects opened in one window, as a workaround you can configure them all in IntelliJ IDEA as modules.

IntelliJ IDEA vs Eclipse terminology

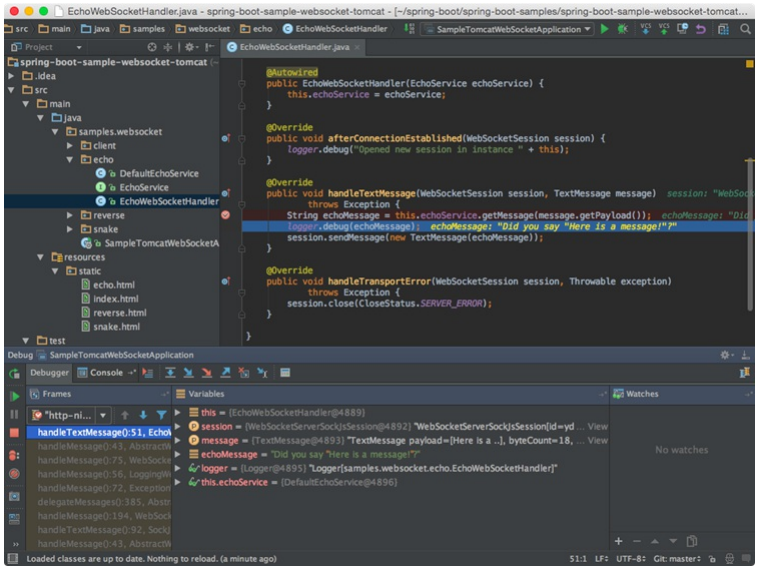
The table below compares the terms in Eclipse and IntelliJ IDEA:

Eclipse	IntelliJ IDEA
Workspace	Project
Project	Module
Facet	Facet
Library	Library
JRE	SDK
Classpath variable	Path variable

No perspectives

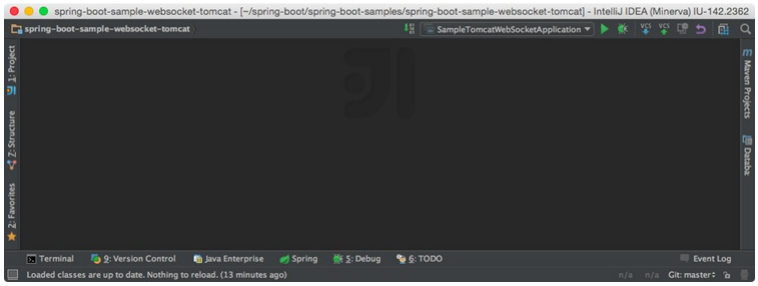
The second big surprise when you switch to IntelliJ IDEA is that it has no perspectives.

It means that you don't need to switch between different workspace layouts manually to perform different tasks. The IDE follows your context and brings up the relevant tools automatically.

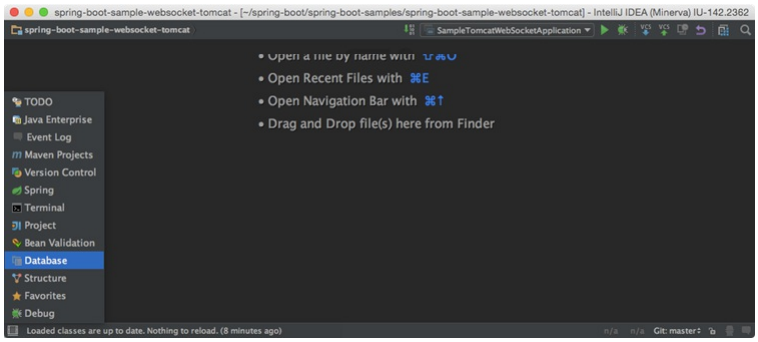


Tool windows

Just like in Eclipse, in IntelliJ IDEA you also have tool windows. To open a tool window, simply click it in the tool window bar:



If the tool window bar is hidden, you can open any tool window by hovering over the corresponding icon in the bottom left corner:

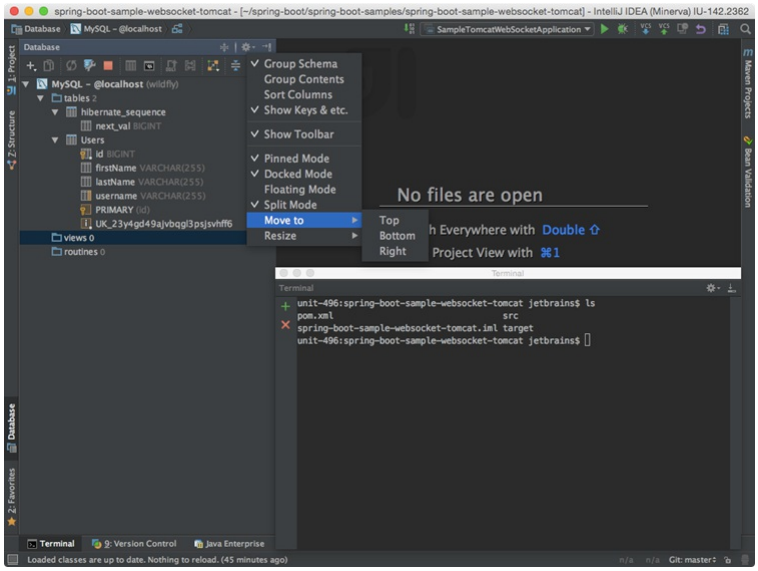


If you want to make the tool window bar visible for a moment, you can press `Alt` (`Cmd` for OS X) twice and hold it.

If you don't want to use the mouse, you can always switch to any toolbar by pressing the shortcut assigned to it. The most important shortcuts to remember are:

- Project: `Alt+1`
- Version Control: `Alt+9`
- Terminal: `Alt+F12`

Another thing about tool windows is that you can drag, pin, unpin, attach and detach them:



To help store/restore the tool windows layout, there are two useful commands:

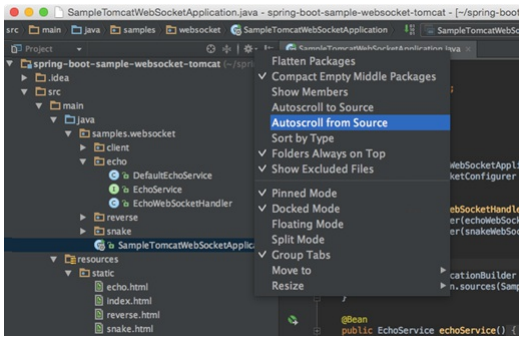
- Window | Store Current Layout as Default
- Window | Restore Default Layout (also available via `Ctrl+F12`)

Multiple windows

Windows management in IntelliJ IDEA is slightly different from Eclipse. You can't open several windows with one project, but you can detach any number of editor tabs into separate windows.

Auto-scrolling to/from sources

By default, IntelliJ IDEA doesn't change the selection in the Project tool window when you switch between editor tabs. However, you can enable it in the tool window settings:



Enabling line numbers

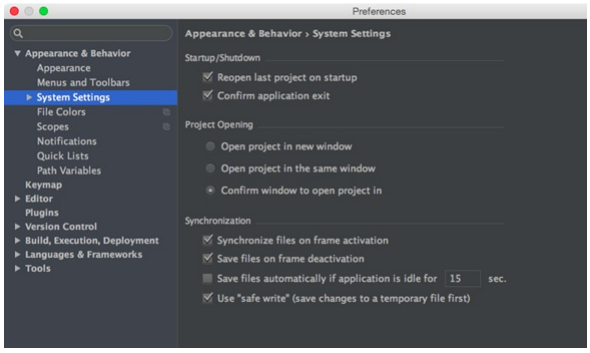
Line numbers are not shown in the editor by default. To enable them, go to Settings/Preferences | Editor | General | Appearance | Show line numbers. There you will also find other useful settings.

General workflows

No 'save' button

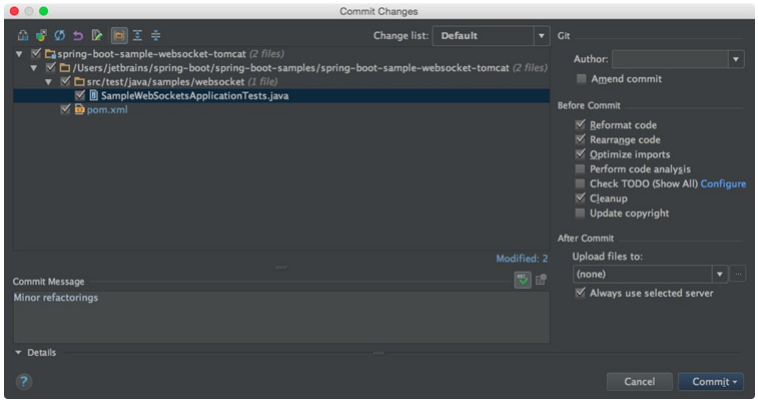
Time for some really shocking news: IntelliJ IDEA has no Save button. Since in IntelliJ IDEA you can undo refactorings and revert changes from Local History , it makes no sense to ask you to save your changes every time.

Still, it's worth knowing that physical saving to disk is triggered by certain events, including compilation, closing a file, switching focus out of the IDE, etc. You can change this behavior via Settings | Appearance & Behavior | System Settings:



No save actions

One of the features you may miss in IntelliJ IDEA as an Eclipse user is save actions, i.e. the actions triggered automatically on save, such as reformatting code, organizing imports, adding missing annotations and the final modifier, etc. Instead, IntelliJ IDEA offers you to run the corresponding actions automatically on commit:



Or manually:

- Code | Reformat Code (`Ctrl+Alt+L`)
- Code | Optimize Imports (`Ctrl+Alt+O`)
- Analyze | Code Cleanup

If, for some reason, you can't live without an Eclipse save action, you can install a plugin that imitates Eclipse save actions .

Compilation

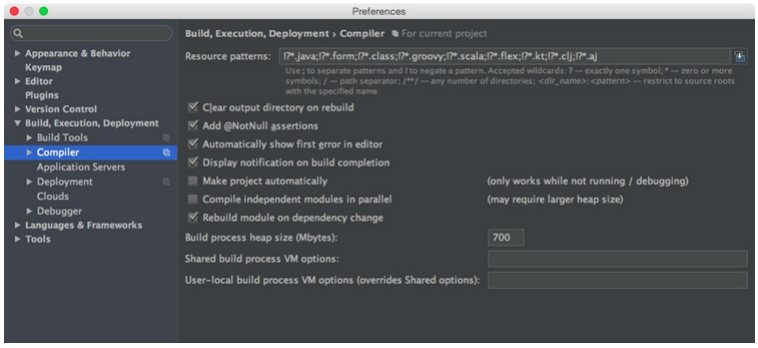
The way IntelliJ IDEA compiles projects is different from Eclipse in a number of ways.

Auto-compilation

By default, IntelliJ IDEA doesn't automatically compile projects on saving because normally we don't invoke the save action explicitly in IntelliJ IDEA.

If you want to mimic the Eclipse behavior, you can invoke the Make Project action (`Ctrl+F9`) - it will save the changed files and compile them. For your convenience, you can even reassign the `Ctrl+S` shortcut to the Make Project action.

To enable automatic compilation, navigate to Settings/Preferences | Build, Execution, Deployment | Compiler and select the Make project automatically option:

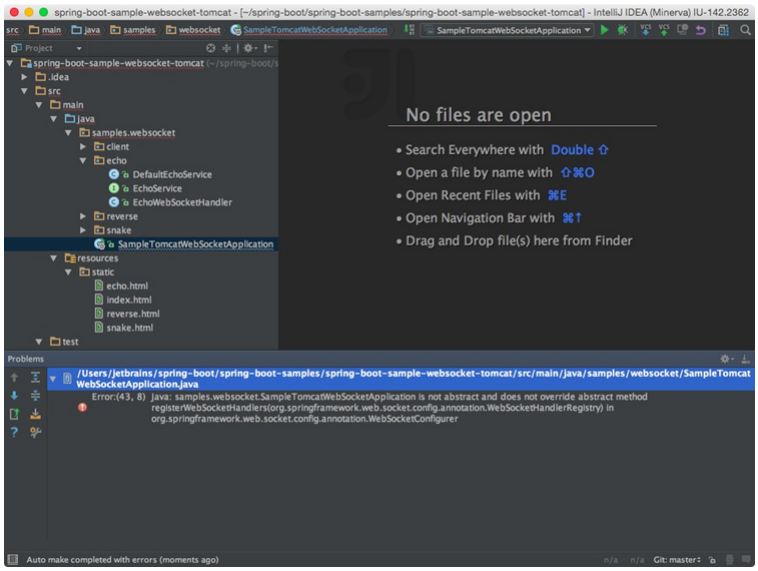


Note that automatic compilation in IntelliJ IDEA differs from that in Eclipse. In Eclipse it's not fully automatic, as it is triggered by the save action invoked by the user explicitly, whereas in IntelliJ IDEA it is invoked implicitly when you type in the editor.

This is why, even if the Make project automatically option is enabled, IntelliJ IDEA doesn't perform automatic compilation if at least one application is running: it will reload classes in the application implicitly. In this case you can call Build | Make Project (`Ctrl+F9`).

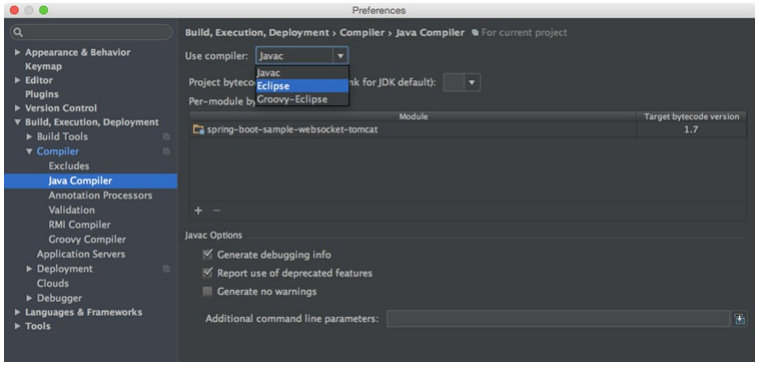
Problems tool window

The Problems tool window appears if the Make project automatically option is enabled in the Compiler settings . It shows a list of problems that were detected on project compilation:



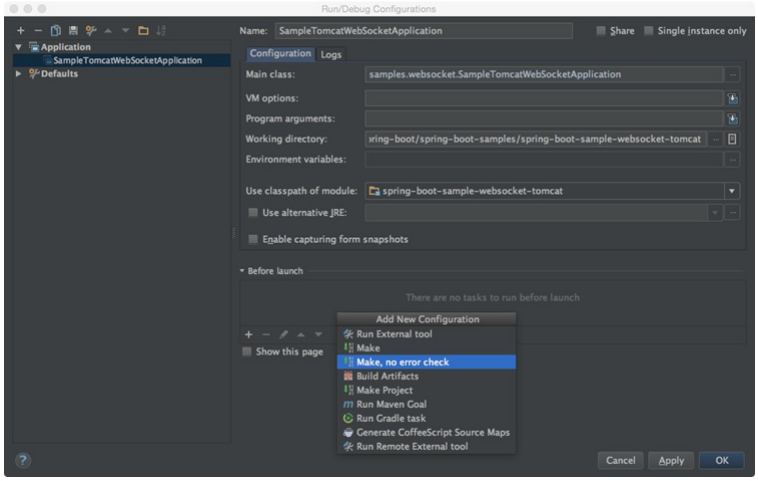
Eclipse compiler

While Eclipse uses its own compiler, IntelliJ IDEA uses the javac compiler bundled with the project JDK. If you must use the Eclipse compiler, navigate to Settings? Preferences | Build, Execution, Deployment | Compiler | Java Compiler and select it as shown below:



The biggest difference between the Eclipse and javac compilers is that the Eclipse compiler is more tolerant to errors, and sometimes lets you run code that doesn't compile.

In situations when you need to run code with compilation errors in IntelliJ IDEA, replace the Make option in your run configuration with Make, no error check:



Shortcuts

IntelliJ IDEA shortcuts are completely different from those in Eclipse.

The table below shows how the top Eclipse actions (and their shortcuts) are mapped to IntelliJ IDEA (you may want to print it out to always have it handy).

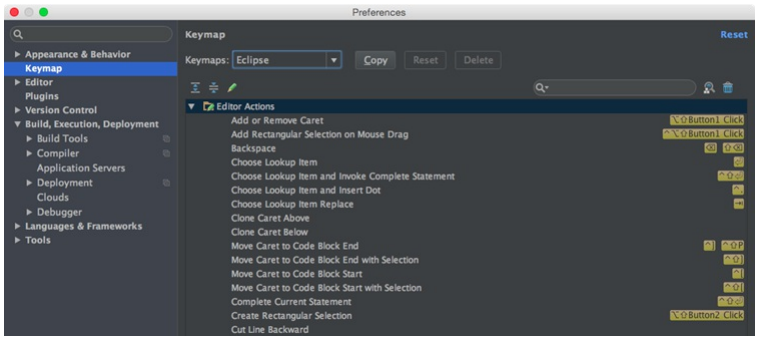
Note If you choose a keymap specific to your operating system (Default for Windows/Linux or OS X 10.5+ for OS X), there may be conflicts between shortcuts used in IntelliJ IDEA and your OS. To avoid such conflicts, we recommend tweaking your OS shortcut settings (refer to Keymap for more details).

Eclipse	**IntelliJ IDEA**		
Action	Shortcut	Action	Shortcut
Code completion	`Ctrl+Space`	Basic completion	`Ctrl+Space`
-	-	Smart completion	`Ctrl+Shift+Space`
-	-	Statement completion	`Ctrl+Shift+Enter`

Quick access	`Ctrl+3`	Search everywhere	`Shift x 2`
Maximize active view or editor	`Ctrl+M`	Hide all tool windows	`Ctrl+Shift+F12`
Open type	`Ctrl+Shift+T`	Navigate to class	`Ctrl+N`
Open resource	`Ctrl+Shift+R`	Navigate to file	`Ctrl+Shift+N`
-	-	Navigate to symbol	`Ctrl+Shift+Alt+N`
Next view	`Ctrl+F7`	-	-
-	-	Recent files	`Ctrl+E`
-	-	Switcher	`Ctrl+Tab`
Quick outline	`Ctrl+O`	File structure	`Ctrl+F12`
Move lines	`Alt+Up/Down`	Move lines	`Shift+Alt+Up` / `Shift+Alt+Down`
Delete lines	`Ctrl+D`	Delete lines	`Ctrl+Y`
Quick fix	`Ctrl+1`	Show intention action	`Alt+Enter`
Quick switch editor	`Ctrl+E`	Switcher	`Ctrl+Tab`
-	-	Recent files	`Ctrl+E`
Quick hierarchy	`Ctrl+T`	Navigate to type hierarchy	`Ctrl+H`
-	-	Navigate to method hierarchy	`Ctrl+Shift+H`
-	-	Show UML popup	`Ctrl+Alt+U`
Last edit location	`Ctrl+Q`	Last edit location	`Ctrl+Shift+Backspace`
Next editor	`Ctrl+F6`	Select next tab	`Alt+Right`
Run	`Ctrl+Shift+F11`	Run	`Shift+F10`
Debug	`Ctrl+F11`	Debug	`Shift+F9`
Correct indentation	`Ctrl+I`	Auto-indent lines	`Ctrl+Alt+I`
Format	`Ctrl+Shift+F`	Reformat code	`Ctrl+Alt+L`
Surround with	`Ctrl+Alt+Z`	Surround with	`Ctrl+Alt+T`
-	-	Surround with live template	`Ctrl+Alt+J`
Open declaration	`F3`	Navigate to declaration	`Ctrl+B`
-	-	Quick definition	`Ctrl+Shift+I`
Open type hierarchy	`F4`	Navigate to type hierarchy	`Ctrl+H`
-	-	Show UML popup	`Ctrl+Alt+U`
References in workspace	`Ctrl+Shift+G`	Find usages	`Alt+F7`
-	-	Show usages	`Ctrl+Alt+F7`
-	-	Find usages settings	`Ctrl+Shift+Alt+F7`
Open search dialog	`Ctrl+H`	Find in path	`Ctrl+Shift+F`
Occurrences in file	`Ctrl+Alt+U`	Highlight usages in file	`Ctrl+Shift+F7`
Copy lines	`Ctrl+Alt+Down`	Duplicate lines	`Ctrl+D`
Extract local variable	`Ctrl+Alt+L`	Extract variable	`Ctrl+Alt+V`
Assign to field	`Ctrl+2` / `Ctrl+F`	Extract field	`Ctrl+Alt+F`
Show refactor quick menu	`Ctrl+Alt+T`	Refactor this	`Ctrl+Shift+Alt+T`
Rename	`Ctrl+Alt+R`	Rename	`Shift+F6`
Go to line	`Ctrl+L`	Navigate to line	`Ctrl+G`
Structured selection	`Shift+Alt+Up` / `Shift+Alt+Down`	Select word at caret	`Ctrl+W` / `Ctrl+Shift+W`
Find next	`Ctrl+J`	Find next	`F3`
Show in	`Ctrl+Alt+W`	Select in	`Alt+F1`
Back	`Ctrl+[`	Back	`Ctrl+Alt+Left`
Forward	`Ctrl+]`	Forward	`Ctrl+Alt+Right`

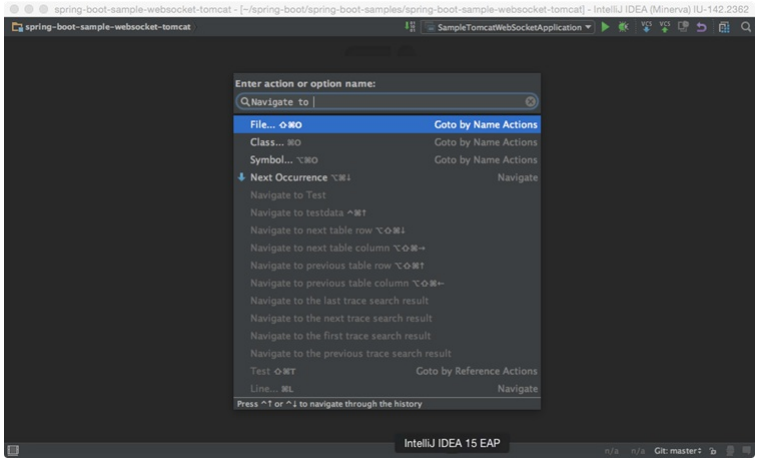
Eclipse keymap

For Eclipse users who prefer not to learn new shortcuts, IntelliJ IDEA provides the Eclipse keymap which closely mimics its shortcuts:



Find action

When you don't know the shortcut for some action, try using the Find action feature available via `Ctrl+Shift+A`. Start typing to find an action by its name, see its shortcut, or call it:

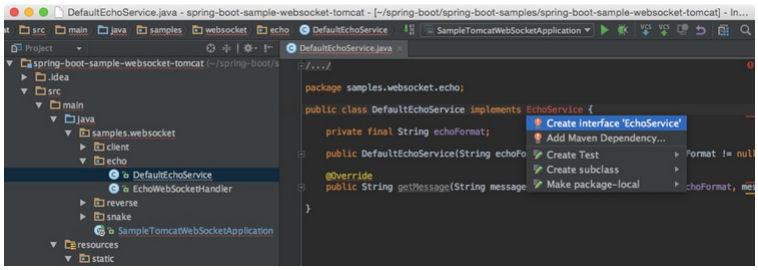


Coding assistance

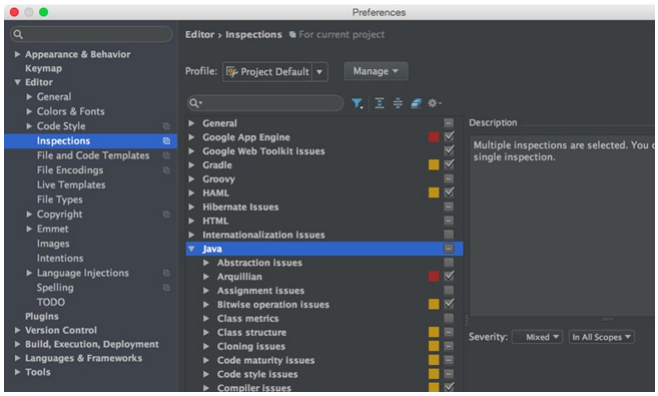
Both Eclipse and IntelliJ IDEA provide coding assistance features, such as code completion, code generation, quick-fixes, live templates, etc.

Quick-fixes

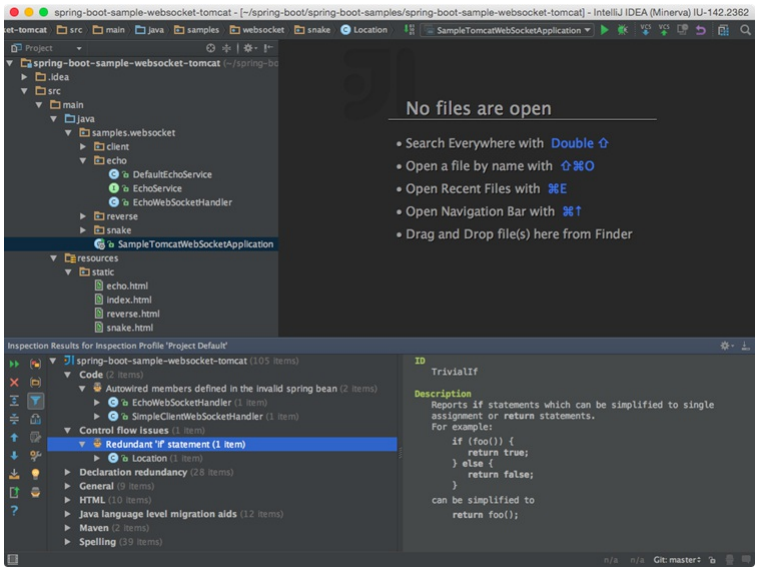
To apply a quick-fix in IntelliJ IDEA, press `Alt+Enter`:



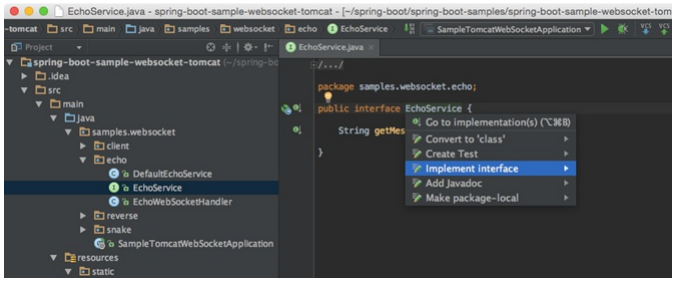
All quick-fixes are based on inspections configured in Settings | Inspections:



If you want to apply a quick-fix to several places at once (i.e. to a whole folder, module or even a project), you can do it by running the corresponding inspection via Analyze | Run Inspection By Name or by running the whole batch of inspections via Analyze | Inspect Code:



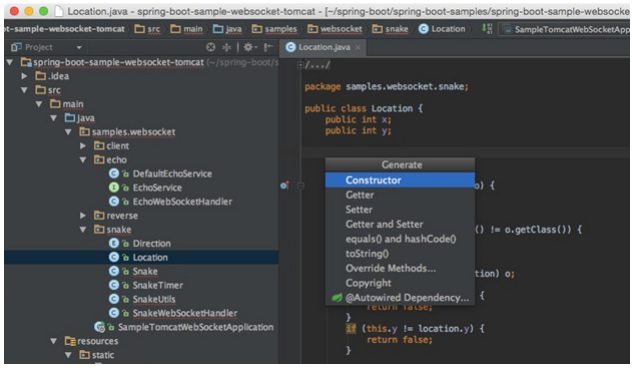
Apart from outright problems, IntelliJ IDEA also recognizes code constructs that can be improved or optimized via the so-called intentions (also available with `Alt+Enter`):



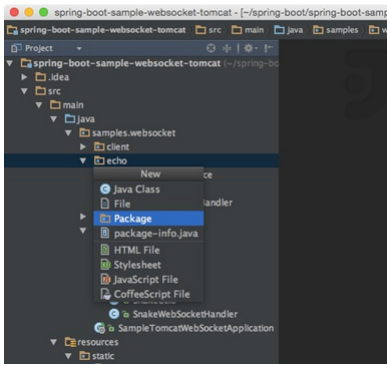
Eclipse	IntelliJ IDEA		
Action	Shortcut	Action	Shortcut
Quick fix	`Ctrl+1`	Show intention action	`Alt+Enter`

Generating code

The key action for generating code is Code | Generate, available via `Alt+Insert`:



This action is context-sensitive and is available not only within the editor, but also in the Project tool window and the Navigation bar :



Code completion

IntelliJ IDEA provides several different types of code completion, which include:

- Basic completion
- Second basic completion

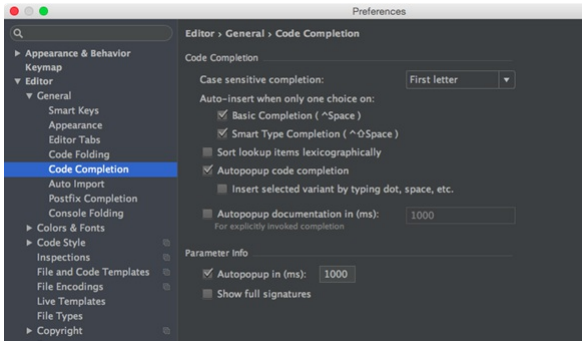
– Smart completion

– Second smart completion

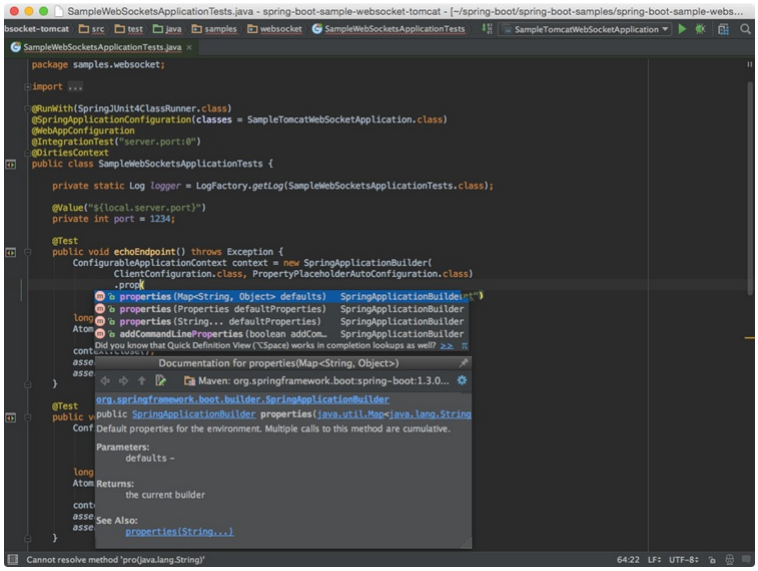
– Statement completion

To learn more about the differences between these completion types, refer to Top 20 Features of Code Completion in IntelliJ IDEA .

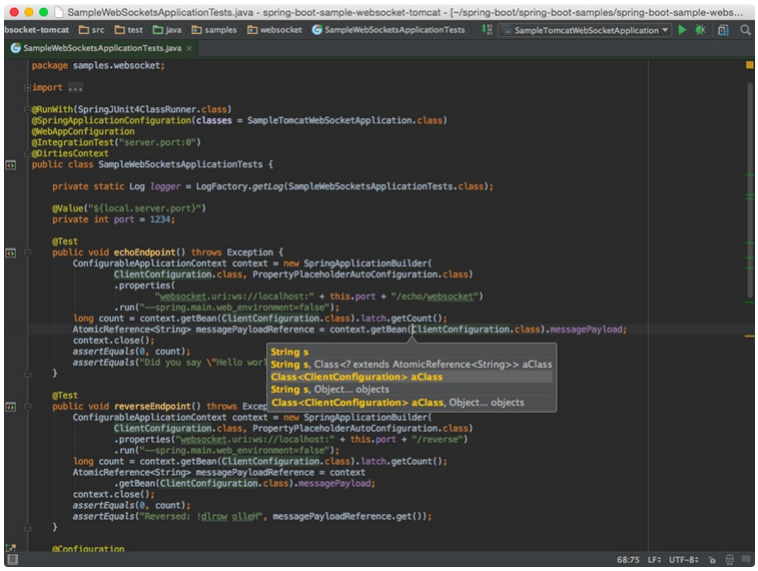
By default, IntelliJ IDEA doesn't show the Documentation popup for the selected item, but you can enable it in Settings/Preferences | Editor | Code Completion | Autopopup documentatoin in (ms):



If you don't want to enable this option, you can manually invoke this popup by pressing `Ctrl+Q` when you need it:



When the caret is within the brackets of a method or a constructor, you can get the info about the parameters by calling Parameter Info with `Ctrl+P`:



Eclipse		IntelliJ IDEA	
Action	Shortcut	Action	Shortcut
Code completion	`Ctrl+Space`	Basic completion	`Ctrl+Space`
-	-	Smart completion	`Ctrl+Shift+Space`
-	-	Statement completion	`Ctrl+Shift+Enter`

Templates

You may be used to typing `main` in the editor and then calling code completion to have it transformed into a main method definition. However, IntelliJ IDEA templates are a little different:

Template	Eclipse	IntelliJ IDEA
Define a main method	`main`	`psvm`
Iterate over an array	`for`	`itar`
Iterate over a collection	`for`	`itco`
Iterate over a list	`for`	`itli`
Iterate over an iterable using foreach syntax	`foreach`	`iter`
Print to System.out	`sysout`	`sout`
Print to System.err	`syserr`	`serr`
Define a static field	`static_final`	`psf`

The list of available templates can be found in Settings/Preferences | Editor | Live Templates. There you can also add your own templates or modify any existing ones.

While IntelliJ IDEA suggests templates in code completion results, you can quckly expand any template without using code completion simply by pressing `Tab`.

Postfix templates

In addition to 'regular' templates, IntelliJ IDEA offers the so-called postfix templates. They are useful when you want to apply a template to an expression you've already typed. For instance, type a variable name, add `.ifn` and press `Tab`. IntelliJ IDEA will turn your expression into a `if (...==null){...}` statement.

To see a complete list of available postfix templates, go to Settings/Preferences | Editor | General | Postfix Completion.

Surround with live template

The surround with templates is another addition that works similarly to live templates but can be applied to the selected code with `Ctrl+Alt+J`.

To define your own surround with template, go to Settings/Preferences | Editor | General | Live Templates and use `$SELECTION$` within the template text:

```
$LOCK$.readLock().lock();
                    try {
                    $SELECTION$
                    } finally {
                    $LOCK$.readLock().unlock();
                    }
```

Navigation

The table below roughly maps the navigation actions available in Eclipse with those in IntelliJ IDEA:

Eclipse		IntelliJ IDEA	
Action	Shortcut	Action	Shortcut
Quick access	`Ctrl+3`	Search everywhere	`Shift x 2`
Open type	`Ctrl+Shift+T`	Navigate to class	`Ctrl+N`
Open resource	`Ctrl+Shift+R`	Navigate to file	`Ctrl+Shift+N`
-	-	Navigate to symbol	`Ctrl+Shift+Alt+N`
Quick switch editor	`Ctrl+E`	Switcher	`Ctrl+Tab`
-	-	Recent files	`Ctrl+E`
Open declaration	`F3`	Navigate to declaration	`Ctrl+B`
Open type hierarchy	`F4`	Navigate to type hierarchy	`Ctrl+H`
-	-	Show UML popup	`Ctrl+Alt+U`
Quick outline	`Ctrl+O`	File structure	`Ctrl+F12`
Back	`Ctrl+[`	Back	`Ctrl+Alt+Left`
Forward	`Ctrl+]`	Forward	`Ctrl+Alt+Right`

Later, when you get used to these navigation options and need more, refer to Top 20 Navigation Features in IntelliJ IDEA.

Refactorings

The following table maps the shortcuts for the most common refactorings in Eclipse with those in IntelliJ IDEA:

Eclipse		IntelliJ IDEA	
Action	Shortcut	Action	Shortcut
Extract local variable	`Ctrl+Alt+L`	Extract variable	`Ctrl+Alt+V`
Assign to field	`Ctrl+2`	Extract field	`Ctrl+Alt+F`
Show refactor quick menu	`Ctrl+Alt+T`	Rafactor this	`Ctrl+Shift+Alt+T`
Rename	`Ctrl+Alt+R`	Rename	`Shift+F6`

To learn more about many additional refactorings that IntelliJ IDEA offers, refer to Top 20 Refactoring Features in IntelliJ IDEA

Undo

Sometimes, refactorings may affect a lot of files in a project. IntelliJ IDEA not only takes care of applying changes safely, but also lets you revert them. To undo the last refactoring, switch the focus to the Project tool window and press `Ctrl+Z`.

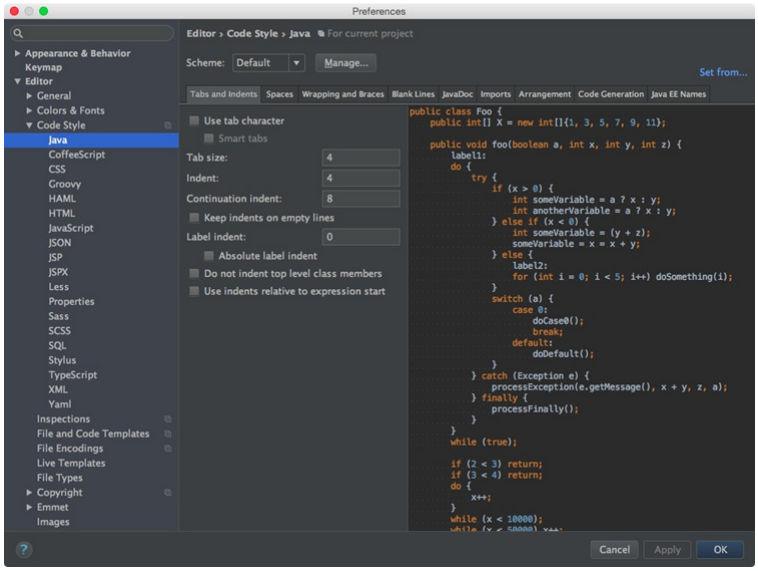
Search

Below is a map of the most common search actions and shortcuts:

Eclipse	IntelliJ IDEA		
Action	Shortcut	Action	Shortcut
Open search dialog	`Ctrl+H`	Find in path	`Ctrl+Shift+F`
References in workspace	`Ctrl+Shift+G`	Find usages	`Alt+F7`
-	-	Show usages	`Ctrl+Alt+F7`
-	-	Find usages settings	`Ctrl+Shift+Alt+F7`
Occurrences in file	`Ctrl+Alt+U`	Highlight usages in file	`Ctrl+F7`

Code formatting

IntelliJ IDEA code formatting rules (available via Settings/Preferences | Editor | Code Style) are similar to those in Eclipse, with some minor differences. You may want to take note of the fact that the Using the Tab char option is disabled by default, the Indent size may be different, etc.



If you would like to import your Eclipse formatter settings, go to Settings/Preferences | Editor | Code Style | Java, click Manage, click Import and select the exported Eclipse formatter settings (an XML file).

> **Tip** Note that if the Eclipse formatter settings cannot be imported, the following error message shows up:
>
> The input file is not a valid Eclipse XML profile.

Note that there may be some discrepancies between the code style settings in IntelliJ IDEA and Eclipse. For example, you cannot tell IntelliJ IDEA to put space after (but not before). If you want IntelliJ IDEA to use the Eclipse formatter, consider installing the Eclipse code formatter plugin.

Eclipse	IntelliJ IDEA		
Action	Shortcut	Action	Shortcut
Format	`Ctrl+Shift+F`	Reformat code	`Ctrl+Alt+L`

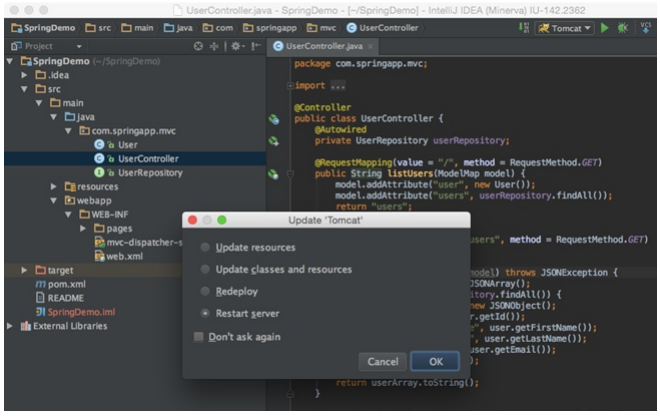
Running and reloading changes

Similarly to Eclipse, IntelliJ IDEA also has Run/Debug Configurations that you can access either form the main toolbar, or the main menu. Compare the related shorcuts:

Eclipse	IntelliJ IDEA		
Action	Shortcut	Action	Shortcut
Run	`Ctrl+Shift+F11`	Run	`Shift+F10`
Debug	`Ctrl+F11`	Debug	`Shift+F9`
-	-	Make	`Ctrl+F9`
-	-	Update application	`Ctrl+F10`

As mentioned before, by default IntelliJ IDEA doesn't compile changed files automatically (unless you configure it to do so). That means the IDE doesn't reload changes automatically. To reload changed classes, call the Make action explicitly via `Ctrl+F9`. If your application is running on a server, in addition to reloading you can use the

Update application action via `Ctrl+F10`:



Debugging

The debuggers in Eclipse and IntelliJ IDEA are similar but use different shortcuts:

Eclipse		IntelliJ IDEA	
Action	Shortcut	Action	Shortcut
Step into	`F5`	Step into	`F7`
-	-	Smart step into	`Shift+F7`
Step over	`F6`	Step over	`F8`
Step out	`F7`	Step out	`Shift+F8`
Resume	`F8`	Resume	`F9`
Toggle breakpoint	`Ctrl+Shift+B`	Toggle breakpoint	`Ctrl+F8`
-	-	Evaluate expression	`Alt+F8`

Working with Application Servers (Tomcat/TomEE, JBoss EAP, Glassfish, WebLogic, WebSphere)

Deploying to application servers in IntelliJ IDEA is more or less similar to what you are probably used to in Eclipse. To deploy your application to a server:

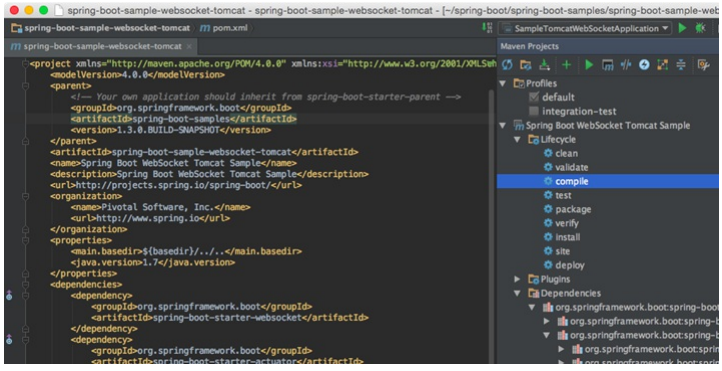
1. Configure your artifacts via Project Structure | Artifacts (done automatically for Maven and Gradle projects).
2. Configure an application server via Settings | Application Servers.
3. Create a run configuration , and then specify the artifacts to deploy and the server to deploy to.

You can always tell the IDE to build/rebuild your artifacts once the have been configured via Build | Build Artifacts.

Working with Build Tools (Maven/Gradle)

IntelliJ IDEA doesn't provide visual forms for editing Maven/Gradle configuration files. Once you've imported/created your Maven/Gradle project, you are free to edit its `pom.xml/build.gradle` files directly in the editor. Later, you can tell IntelliJ IDEA to synchronize the project model with the changed files on demand, or automatically import changes to the new build files. Any changes to the underlying build configuration will eventually need to be synced with the project model in IntelliJ IDEA.

For operations specific to Maven/Gradle, IntelliJ IDEA provides the Maven Project tool window and the Gradle tool window . Apart from your project structure, these tool windows provide a list of goals/tasks plus a toolbar with the relevant actions.



If you want the IDE to synchronize your changes immediately:

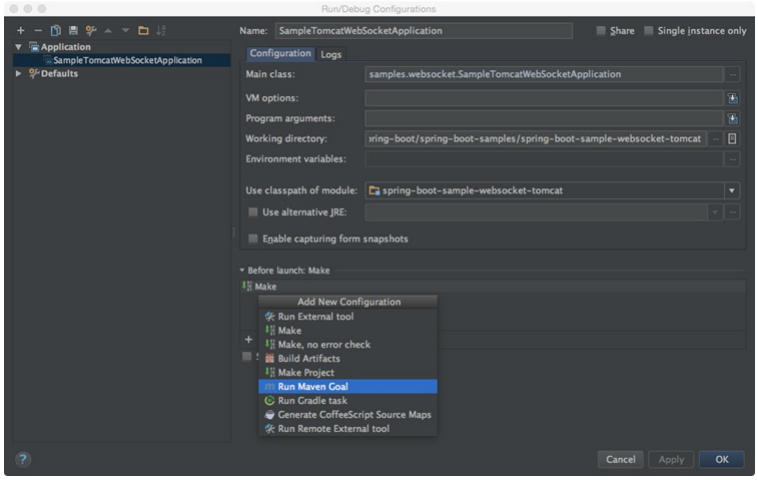
- For `pom.xml` , enable the corresponding options in Settings | Build, Execution, Deployment | Build Tools | Maven | Importing | Import Maven projects automatically
- For `build.gradle` , enable the corresponding option in Settings | Build, Execution, Deployment | Build Tools | Gradle | Use auto-import .

For manual synchronization, use the corresponding action on the Maven/Gradle tool window toolbar: .

Running goals/tasks

Use the Maven/Gradle tool window to run any project goal/task. When you do, IntelliJ IDEA creates the corresponding run configuration which you can reuse later to run the goal/task quickly.

It's worth mentioning that any goal/task can be attached to be run before a Run Configuration. This may be useful when your goal/task generates specific files needed by the application.



Both the Maven and Gradle tool windows provide the Run Task action. It runs a Maven/Gradle command similarly to how you'd run it using the console.

Configuring artifacts

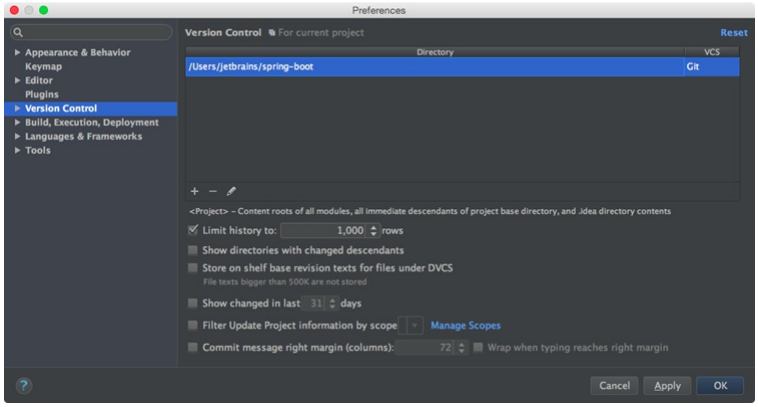
If you have WAR artifacts configured in your `pom.xml/build.gradle` file, IntelliJ IDEA automatically configures the corresponding artifacts in Project Structure | Artifacts.

Note that when you compile your project or build an artifact, IntelliJ IDEA uses its own build process which may be faster, but is not guaranteed to be 100% accurate. If you notice inconsistent results when compiling your project with Make in IntelliJ IDEA, try using a Maven goal or a Gradle task instead.

Working with VCS (Git, Mercurial, Subversion, Perforce)

Configuring VCS roots

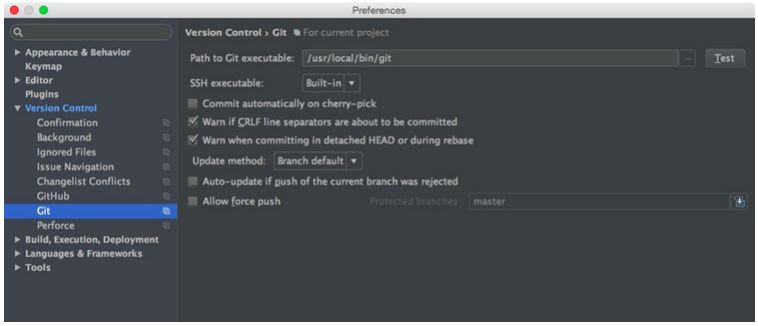
When you open a project located under a VCS root, IntelliJ IDEA automatically detects it and suggests adding this root to the project settings. To change version control-related project settings (or manually add a VCS root), go to Settings | Version Control:



IntelliJ IDEA works perfectly with multi-repository projects. Just map your project directories to VCS, and the IDE will take care of the rest. For Git and Mercurial , the IDE will even offer you synchronized branch control, so that you can perform branch operations on multiple repositories simultaneously (for more details, see Git Branches in Multirooted Projects).

Editing VCS settings

Every VCS may require specific settings, for example, Path to Git executable, GitHub/Perforce credentials, etc.:



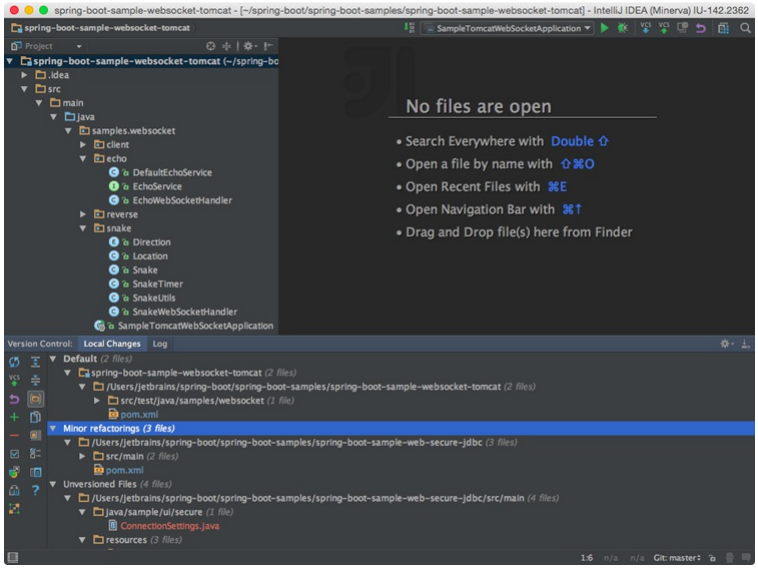
Once you've configured the VCS settings, you'll see the Version Control tool window . You can invoke it any time by pressing `Alt+9`.

Checking projects out

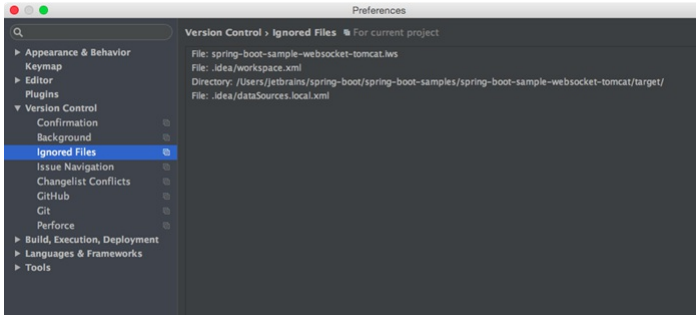
To check out a project from a VCS, click Checkout from Version Control on the Welcome Screen , or in the main VCS menu.

Working with local changes

The Local Changes tab of the Version Control tool window shows your local changes: both staged and unstaged. To simplify managing changes, all changes are organized into changelists. Any changes made to source files are automatically included into the active changelist. You can create new changelists, delete the existing ones (except for the Default changelist), and move files between changelists.

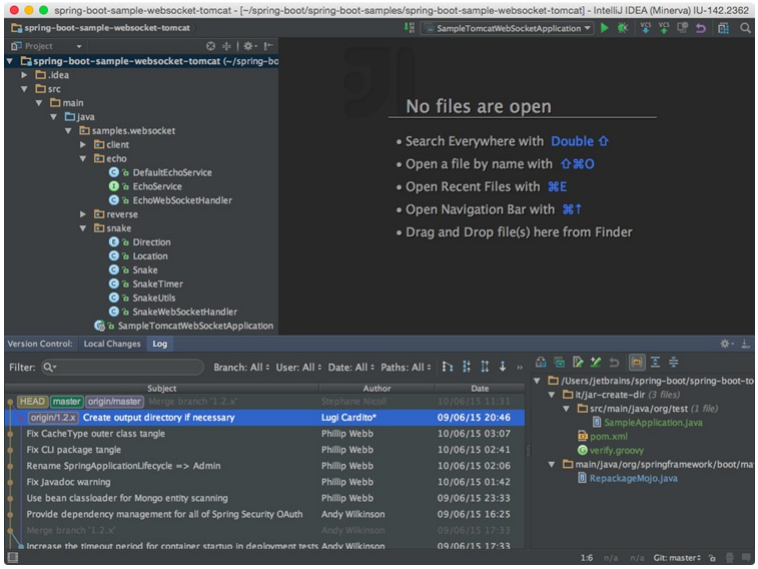


To configure ignored files, go to Settings | Version Control, or use the corresponding button in the Version Control tool window.



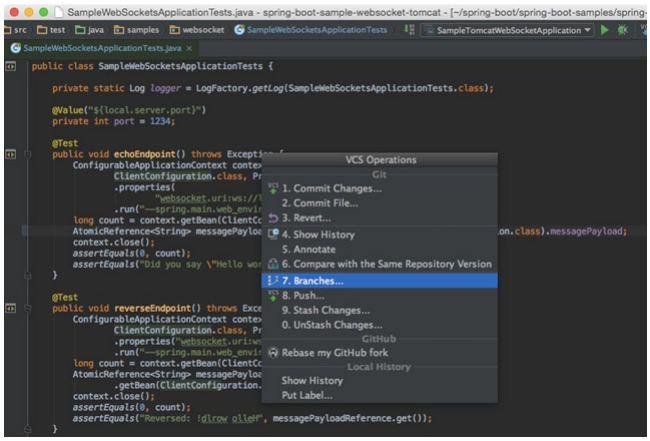
Working with history

The Log tab of the Version Control tool window lets you see and search through the history of commits. You can sort and filter commits by the repository, branch, user, date, folder, or even a phrase in the description. You can find a particular commit, or just browse through the history and the branch tree:



Working with branches

IntelliJ IDEA lets you create, switch, merge, compare and delete branches. For these operations, either use Branches from the main or context VCS menu, or the VCS operations popup (you can invoke it by pressing `Alt+Back Quote`, or the widget on the right of the status bar:



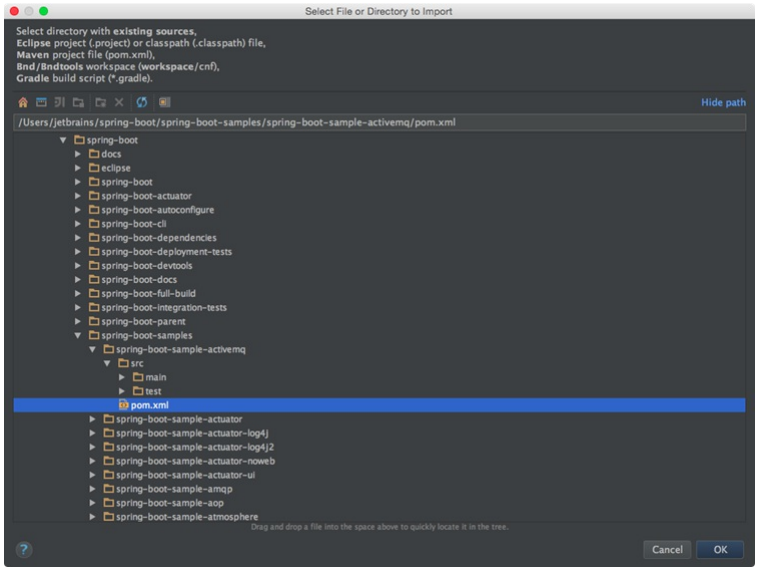
All VCS operations are available from the VCS main menu:

Action	Shortcut
Version Control tool window	`Alt+9`
VCS operations popup	`Alt+Back Quote`
Commit changes	`Ctrl+K`
Update project	`Ctrl+T`
Push commits	`Ctrl+Shift+K`

Importing an Eclipse project to IntelliJ IDEA

Despite these differences in terms and the UI, you can import either an Eclipse workspace or a single Eclipse project. To do this, click Import Project on the Welcome Screen, or select File | New | Project from Existing Sources in the main menu.

If your project uses a build tool such as Maven or Gradle , we recommend choosing the corresponding option when prompted in the Import Project wizard, and selecting the associated build file (`pom.xml` or `build.gradle`):



If you'd like to import your existing run configurations from Eclipse, consider using this third-party plugin .

Exporting an IntelliJ IDEA Project to Eclipse

On this page:

– Overview
– Exporting to Eclipse
– Converting an IntelliJ IDEA module to the Eclipse-compatible format

Overview

You can export an IntelliJ IDEA project to Eclipse. Such export results in creating Eclipse project files (`.project` and `.classpath`) for each module file (`*.iml`) in the module directory that contains the content root. Another way to export an IntelliJ IDEA module to an Eclipse project is converting such module to an Eclipse-compatible format.

Before you start exporting a project, make sure that the Eclipse Integration plugin is enabled .

Exporting to Eclipse

To export the currently open project to Eclipse, follow these steps:

1. On the main menu, choose File | Export to Eclipse. The Export to Eclipse dialog displays the list of modules that have not been converted and switched to use the Eclipse format yet (the modules that have the IntelliJ IDEA module format `.iml`).
2. Select the modules you want to export.
3. Select the suggested options if necessary.
4. Click OK.

Note At present, migration from IntelliJ IDEA to Eclipse is subject to certain limitations:
– IntelliJ IDEA modules with multiple content roots cannot not be migrated.
– External sources in Eclipse are not migrated.
– Only Java modules are created automatically during the export. However, any module can be converted to the Eclipse compatible format manually.
– The default workspace JRE is not converted to/from the project SDK.

Converting an IntelliJ IDEA module to the Eclipse-compatible format

To convert an IntelliJ IDEA module to the Eclipse-compatible format, follow these steps:

1. On the main menu, choose File | Project Structure, or press `Ctrl+Shift+Alt+S` .
2. In the Project Structure Dialog dialog, select the module you want to convert.
3. Switch to the Dependencies Tab tab.
4. From the Dependencies storage format drop-down list, select Eclipse (.classpath).

NetBeans

NetBeans users ask the following question about IntelliJ IDEA most frequently:

- ABOUT PROJECTS
  - How do I open a NetBeans project in IntelliJ IDEA?
  - What's the difference between projects and modules?
  - Is there a directory-based project format in IntelliJ IDEA?
  - How do I change the JDK for my project?
  - How do I add a library to my project?
  - How do I configure a Web framework for my project?
  - The Run button is disabled. How do I run my application?
  - How do I generate an Ant build script for my project?
  - Where is the Options dialog?
  - How do I close a project?
- ABOUT THE EDITOR
  - Can I use the NetBeans key bindings in IntelliJ IDEA?
  - How does code completion in IntelliJ IDEA work?
  - Is local history in IntelliJ IDEA any different from that in NetBeans?
  - Are there any special code analysis features in IntelliJ IDEA?
  - Can I enable 'mark occurrences' in IntelliJ IDEA?
  - Can I enable 'compile on save' in IntelliJ IDEA?
  - Can I enable 'deploy on save' in IntelliJ IDEA?
- ABOUT PLUGINS
  - Can I use NetBeans plugins in IntelliJ IDEA?
  - How do I find the plugin that I need?
  - How do I install the plugin that I have available on my computer?
  - I'd like to write a plugin for IntelliJ IDEA. Are there any instructions?
  - Is it possible to build NetBeans RCP applications with IntelliJ IDEA?

ABOUT PROJECTS

How do I open a NetBeans project in IntelliJ IDEA?

Use File | New | Project from Existing Sources and select your NetBeans project directory.

When the Import Project wizard opens, select the Create project from existing sources option and then follow the instructions of the wizard.

IntelliJ IDEA will add the necessary definition files (the `.idea` directory) to your project directory. The NetBeans `.nbproject` directory and `build.xml` will remain untouched, and you'll be able to use IntelliJ IDEA along with NetBeans.

During the import IntelliJ IDEA will fix missing libraries, add facets for different Web frameworks and create a run configuration.

If you are using Maven with NetBeans and you want to import a Maven project into IntelliJ IDEA, select File | Open and then select your project's `pom.xml`. You'll still need to configure a run configuration, however, all project dependencies will get resolved.

See also, Creating a Project by Importing Existing Sources .

What's the difference between projects and modules?

IntelliJ IDEA creates a project for an entire code base and a module for each of its individual components. So, IntelliJ IDEA module is more like a NetBeans project.

The following table maps the most important NetBeans concepts to IntelliJ IDEA ones.

NetBeans	IntelliJ IDEA
Project	Module
Global library	Global library
Project library	Module library
Project dependency	Module dependency

Is there a directory-based project format in IntelliJ IDEA?

Yes, there is a `.idea` directory where project definition XML files are stored. For more information, see Project .

How do I change the JDK for my project?

1. Open the Project Structure dialog (File | Project Structure or `Ctrl+Shift+Alt+S`).
2. Under Platform Settings, select SDKs.
3. Click +, select JDK and specify the JDK installation directory.
4. Click Apply.
5. Under Project Settings, select Project.
6. Under Project SDK, select the JDK from the list.
7. Click OK.

For more information, see Configuring Global, Project and Module SDKs .

How do I add a library to my project?

1. Open the Project Structure dialog (File | Project Structure or `Ctrl+Shift+Alt+S`).
2. Under Project Settings, select Libraries.
3. Click +, select Java and specify the library location.
4. Select the modules in which this library will be used.
5. Click OK.

For more information, see Configuring Project and Global Libraries and Configuring Module Dependencies and Libraries .

How do I configure a Web framework for my project?

In IntelliJ IDEA Ultimate (there's no corresponding functionality in the Community Edition):

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Right-click the necessary module and select Add Framework Support. (Framework support is enabled at a module level.)
3. In the Add Frameworks Support dialog that opens, select the frameworks to be supported, specify the associated settings and click OK.

For more information, see Add Frameworks Support Dialog and e.g. Enabling JSF support for an existing module.

The Run button is disabled. How do I run my application?

The Run button is disabled because there are no run configurations in your project.

If you have a Java class with a `main()` method, open the corresponding file in the editor, right-click the editing area and select Run '<FileName>.main()'. As a result, the necessary run configuration will be created automatically and then executed.

You can also create run configurations yourself: e.g. Run | Edit Configurations | +, etc.

How do I generate an Ant build script for my project?

Select Build | Generate Ant Build. For more information, see Generating Ant Build File .

Where is the Options dialog?

In IntelliJ IDEA, the Settings dialog is used for similar purposes. To open this dialog, press `Ctrl+Alt+S` .

There is also the Project Structure dialog (`Ctrl+Shift+Alt+S`) which lets you manage JDKs, libraries, module dependencies, etc.

For more information, see Settings / Preferences Dialog and Project Structure Dialog .

How do I close a project?

Select File | Close Project. You can also use File | Exit to close all open projects and quit IntelliJ IDEA. See also, Opening, Reopening and Closing Projects .

ABOUT THE EDITOR

Can I use the NetBeans key bindings in IntelliJ IDEA?

Yes, you can.

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the Appearance and Behavior category, select Keymap.
3. In the right-hand part of the dialog, next to Keymaps, select NetBeans 6.5 from the list.
4. Click OK.

How does code completion in IntelliJ IDEA work?

The code completion suggestion list appears automatically after you type one or two letters. To narrow down this list, use:

- `Ctrl+Space` . The list is reduced to keywords and also the names of classes, methods and fields available in the current context. Note that the list changes when you press `Ctrl+Space` for the second or third time.
- `Ctrl+Shift+Space` . Only the types appropriate for the current context are shown.

For more information, see Auto-Completing Code .

Is local history in IntelliJ IDEA any different from that in NetBeans?

Local history in IntelliJ IDEA, generally, is more detailed. Whatever you do with a directory, file, class, method or field, or a code block is reflected in your local history. The local history also includes VCS operations.

For more information, see Using Local History .

Are there any special code analysis features in IntelliJ IDEA?

IntelliJ IDEA can analyze dependencies, data flows and stacktraces, find duplicates and evaluate code quality. Just have a look at the options in the Analyze menu.

For more information, see Analyzing Applications and Code Inspection .

Can I enable 'mark occurrences' in IntelliJ IDEA?

You can. The corresponding option in IntelliJ IDEA is called Highlight usages of element at caret. This option is enabled by default.

Just in case:

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the Editor category, select General.
3. In the right-hand part of the dialog, under Highlight on Caret Movement, select the Highlight usages of element at caret check box.

4. Click OK.

See also, Highlighting Usages .

Can I enable 'compile on save' in IntelliJ IDEA?

You can.

To enable automatic compilation on every save (or autosave), turn on the Make project automatically option in the Settings dialog:

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the Build, Execution, Deployment category, select Compiler.
3. In the right-hand part of the dialog, select the Make project automatically check box.
4. Click OK.

Also note that by default, IntelliJ IDEA saves changed files automatically, so you don't need to use `Ctrl+S` as frequently as in other IDEs.

Can I enable 'deploy on save' in IntelliJ IDEA?

There is no such option in IntelliJ IDEA settings, however, you can get similar result by choosing an appropriate application update option in the corresponding run configuration.

For more information, see Updating Applications on Application Servers .

(The corresponding functionality is available only in IntelliJ IDEA Ultimate. The Community Edition doesn't provide integration with application servers.)

ABOUT PLUGINS

Can I use NetBeans plugins in IntelliJ IDEA?

Unfortunately not. However, a lot of functionality implemented as plugins for NetBeans is available in IntelliJ IDEA "out of the box". Besides, there's a lot of plugins for IntelliJ IDEA, so you can always find an IntelliJ IDEA plugin with the functionality similar to that of your favorite NetBeans plugin.

How do I find the plugin that I need?

All the functions related to working with plugins are on the Plugins page of the Settings dialog (`Ctrl+Alt+S` | Plugins). You can look for, download, install and update the plugins as well as enable and disable them.

For more information, see Plugins and Managing Plugins .

How do I install the plugin that I have available on my computer?

1. Open the Settings dialog (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, select Plugins.
3. In the lower part of the Plugins page, click Install plugin from disk.
4. In the dialog that opens, select the plugin file (normally, a JAR or ZIP).
5. Click OK.
6. If asked, restart IntelliJ IDEA.

I'd like to write a plugin for IntelliJ IDEA. Are there any instructions?

Yes, have a look at:

- Plugin Development Guidelines
- Information for Plugin Developers on the IntelliJ IDEA Plugins page
- Open API and Plugin Development

Is it possible to build NetBeans RCP applications with IntelliJ IDEA?

It is possible, however you won't get the same kind of support you would in the case of NetBeans (wizards, menu actions, etc.). Have a look at Using IntelliJ IDEA for NetBeans Platform Development .

IntelliJ IDEA vs NetBeans Terminology

NetBeans and IntelliJ IDEA use different names for similar entities. It is important to understand the difference between the terminologies of both IDEs. In this section you can find a table of mappings between the NetBeans and IntelliJ IDEA terms.

NetBeans	IntelliJ IDEA
Project	Module
Project-specific JDK	Module SDK
Global library	Global library
Project library	Module library
Project dependency	Module dependency

Reference

This part contains miscellaneous information related to the UI of the dialogs, tool windows and views, keyboard shortcuts, syntax references etc.:

- Essentials
- New Project Wizard
- New Module Wizard
- Tool Windows Reference
- Import Project or Module Wizard
- Dialogs
- Settings / Preferences Dialog
- Project Structure Dialog
- Keyboard Shortcuts and Mouse Reference
- Version Control Reference
- Java EE Reference
- Table Editor
- Android Reference
- Flex Reference
- Diagram Reference
- GUI Designer Reference
- Icons Reference
- Regular Expression Syntax Reference
- Scope Language Syntax Reference
- Index of Menu Items
- Tuning IntelliJ IDEA

Essentials

IntelliJ IDEA is an advanced IDE. To get the most out of its capabilities and features, you should be familiar with its concepts. Concepts describe the basic notions of the IDE.

In this part:

– Project
– SDK
– Module
– Facet
– Library
– Path Variables
– Supported Languages
– Scope
– Encoding
– Code Analysis

Project

On this page:

– Basics
– Project format and files
– Directory-based format
– File-based format

Basics

Whatever you do in IntelliJ IDEA, you do that in the context of a project. A project is an organizational unit that represents a complete software solution. It serves as a basis for coding assistance, bulk refactoring, coding style consistency, etc.

Your finished product may be decomposed into a series of discrete, isolated modules, but it's a project definition that brings them together and ties them into a greater whole.

Projects don't themselves contain development artifacts such as source code, build scripts, or documentation. They are the highest level of organization in the IDE, and they define project-wide settings as well as collections of what IntelliJ IDEA refers to as modules and libraries .

Project format and files

IntelliJ IDEA stores the configuration data for projects and their components in plain text XML files making it easy to manage and share project configuration data with others.

Two formats are available for storing the project configuration data: directory-based and file-based .

Directory-based format

When the directory-based format is used, there is a `.idea` directory in the project directory.

The `.idea` directory contains a set of configuration files (`.xml`). Each file contains only a portion of configuration data pertaining to a certain functional area which is reflected in the name of a file, for example, `compiler.xml` , `encodings.xml` , `modules.xml` .

Almost all of the files contain information core to the project itself, such as names and locations of its component modules, compiler settings, etc. Thus, these files may (and should) be kept under version control.

The exception is the file `workspace.xml` . It stores your personal settings such as placement and positions of your windows, your VCS and History settings, and other data pertaining to the development environment. So it's unlikely that you'd want to share this file with your colleagues.

File-based format

If the file-based format is used, there are two configuration files in the project directory. One of the files has the `.ipr` extension and stores the core project information. The other file has the `.iws` extension and stores your personal workspace settings.

The `.iws` file shouldn't be placed under version control while the `.ipr` file should.

The file-based format can be converted into the directory-based, see Converting Projects Into Directory-Based Format .

SDK

To develop applications, you need a Software Development Kit (SDK). A typical SDK example is the Java Development Kit (also known as Java SDK, or JDK). This is an SDK for producing Java programs.

IntelliJ IDEA does not include an SDK. So, before you start writing your code, you have to download and install at least one SDK and define it in IntelliJ IDEA. (You can specify an SDK when creating your first project or at a later time.)

To define an SDK in IntelliJ IDEA, generally, all you have to do is to specify its name and location. (This location, normally, is referred to as the SDK home directory. This is the directory in which the SDK is installed). Additional SDK adjustments are possible, however, in most of the cases, they are not necessary.

- Supported SDKs
- SDK levels
- SDK classpath, sourcepath, and documentation paths

Supported SDKs

Depending on the nature of your application, you can use:

- Java SDKs to develop Java desktop applications.
- Mobile SDKs (Java Micro Edition (ME) SDKs) to develop applications for Java-enabled mobile devices.
- Android SDKs to develop applications for Android devices.
- Flex and AIR SDKs to develop applications for the Adobe Flash platform .
- Flexmojos SDKs to launch the Flex compiler and debugger. (Such SDKs are created by IntelliJ IDEA automatically when importing Flexmojos projects.)
- IntelliJ Platform Plugin SDKs to develop plugins for IntelliJ IDEA. (An IntelliJ IDEA installation acts as an IntelliJ Platform Plugin SDK.)

Note that the set of available SDK types depends on the IntelliJ IDEA edition being used (Community or Ultimate) and also on which plugins are currently enabled.

SDK levels

SDKs, generally, are defined at the global (i.e., the IDE) level and can be used in any of your projects .

One of the SDKs is assigned to the project level and is referred to as a project SDK. This SDK is used by default by all the modules of the corresponding project.

A module can use ("inherit") the project SDK, or, alternatively, a module-specific SDK can be specified.

SDK classpath, sourcepath, and documentation paths

When you add an SDK definition in IntelliJ IDEA, the IDE analyses the contents of the SDK installation folder (home directory) to find out where the corresponding binaries, sources and associated documentation are located. As a result, all the appropriate locations within the SDK are translated into corresponding paths and assigned to one of the following categories:

- Classpath. This is a set of paths to directories and archives containing the SDK binaries. For a JDK, for example, this set includes the paths to class directories and JAR files that make up the Java API and its extensions.
  You can add more SDK libraries to this set, however, this is never necessary for any standard SDK.
- Sourcepath. This is a set of paths to directories and archives containing the source code for the SDK binaries included in the classpath set.
  Including the source files isn't required, but doing so improves IntelliJ IDEA's ability to provide inline help while coding.
- Documentation paths. This is a set of paths to documentation that describes the SDK source code.
  Including documentation paths is also optional, but doing so gives you the ability to view documentation directly from the IDE.

  If online documentation is available for the SDK, you can add the corresponding URL to the SDK documentation paths to be able to access this documentation when needed.

Module

A module is a discrete unit of functionality which you can compile, run, test and debug independently.

Modules contain everything that is required for their specific tasks: source code, build scripts, unit tests, deployment descriptors, and documentation. However, modules exist and are functional only in the context of a project .

Configuration information for a module is stored in a `.iml` module file. By default, such a file is located in the module's content root folder .

Development teams, normally, share the `.iml` module files through version control.

In this section:

- Supported Module Types
- Content Root
- Dependencies

Supported Module Types

Supported module types are listed and briefly explained in the following table. Note that the set of available module types depends on the IntelliJ IDEA edition being used (Community or Ultimate) and also on which plugins are currently enabled.

Module type	Description	Editions
Java Module	Encapsulates core functionality for building Java applications. The functionality of a Java module can be expanded by adding facets .	Community and Ultimate
Web Module	Provides facilities for developing web applications using programming languages other than Java, for example, PHP , or JavaScript , or markup languages . Warning! Do not confuse this module with the web folder which IntelliJ IDEA creates when you enable Java-based Web development support .	Ultimate
Plugin Module	Facilitates development of IntelliJ IDEA plugins . Supports IntelliJ IDEA SDK configuration and run configurations for running a dedicated IntelliJ IDEA instance for plugin debugging.	Community and Ultimate
J2ME Module	Provides facilities for developing J2ME mobile applications , including support for various Mobile JDK and J2ME-oriented run configurations.	Ultimate
Flash Module	This module type is intended for developing Adobe Flash platform-targeted content (applications and libraries for web (Flash Player), desktop (Adobe AIR) and mobile devices (AIR mobile)).	Ultimate
Android Module	Encapsulates core functionality for developing Android applications .	Community and Ultimate
Maven Module	Provides facilities for managing Maven projects, executing Maven goals, downloading artifacts.	Community and Ultimate
Grails Application	Provides facilities for Grails application development .	Ultimate
Griffon Application	Provides facilities for Griffon application development .	Community and Ultimate
The following module types are available provided that the corresponding plugins are downloaded and enabled :		
Ruby Module	Provides facilities for creating Ruby projects and Rails applications. Visit Ruby plugin homepage .	Ultimate
Python Module	Provides facilities for Python, Django and Google App Engine development. Visit the Python plugin homepage .	Ultimate

Content Root

On this page:

- Basics
- Folder categories
- Modules without content roots: Collections of dependencies
- Package prefix for Java source roots

See also, Configuring Content Roots .

Basics

Content root is a folder that contains all the files that make up your module.

A module can have more than one content root, however, in most of the cases one content root will suffice. (In certain situations, the modules without content roots may be useful.)

Content roots in IntelliJ IDEA are shown as or .

Folder categories

Folders within a content root can be assigned to the following categories:

- Source roots (or source folders; shown as).
  By assigning a folder to this category, you tell IntelliJ IDEA that this folder and its subfolders contain source code that should be compiled as part of the build process.

  In Java modules, the subfolders within the source roots represent your Java package structure.

  You can assign a package prefix to a source root (e.g. `com.mycompany.myapp`) instead of creating the corresponding folder structure within that source root (e.g. `com/mycompany/myapp`). For more information, see Package prefix for Java source roots.

  In the absence of the package prefix and the subfolders, a source root would represent the default package (an unnamed package).
- Generated source roots (or generated source folders; shown as ; in certain views the [generated] text marker is used) are similar to source roots. The difference is that the generated source roots are not suggested as target folders when performing the Move Class refactoring or using the Create Class from Usage quick fix .
- Test source roots (or test source folders; shown as).
  These roots are similar to source roots but are for code intended for testing (e.g. for unit tests). Test source folders let you keep the code related to testing separate from the production code.

  Compilation results for sources and test sources, normally, are placed into different folders.
- Generated test source roots (or generated test source folders; shown as ; in certain views the [generated] text marker is used) are similar to test source roots. The difference is the same as between the source roots and the generated source roots.
- Resource roots (or resource folders; shown as ; available only in Java modules) are for resource files used in your application (images, various configuration XML and properties files, etc.).
  During the build process, all the contents of the resource folders are copied to the output folder as is.

  Similarly to sources, you can specify that your resources are generated. You can also specify which folder within the output folder your resources should be copied to.
- Test resource roots (or test resource folders; shown as ; available only in Java modules) are for resource files associated with your test sources. In all other respects, these folders are similar to resource folders.
- Excluded roots (shown as) are ones that IntelliJ IDEA "almost ignores".
  Very limited coding assistance is provided for files in excluded folders. Classes contained in excluded folders don't appear in code completion suggestion lists, references to such classes are shown in the editor as unresolved. When searching, IntelliJ IDEA doesn't look in excluded folders, etc.

  Making the folders whose contents are "less important" excluded may improve the IDE performance.

  Normally excluded are compilation output folders.

Modules without content roots: Collections of dependencies

A module can be used solely as a collection of dependencies for other modules. In such cases, instead of specifying the necessary dependencies separately, you can add the dependency on the corresponding module.

A module used for such purpose, obviously, doesn't need a content root.

Package prefix for Java source roots

A package prefix specifies which Java package corresponds to a folder and can be assigned to the following categories of the Java source roots:

- Sources
- Generated sources
- Test sources
- Generated test sources

If specified, the package prefix acts as an equivalent of the corresponding folder structure which has to be created otherwise.

To illustrate, let's assume you are going to work with the `com.mycompany.myapp` package. In the absence of the package prefix, you would create the folder structure `com/mycompany/myapp` in the corresponding source root folder (e.g. `src`).

The alternative would be to assign `com.mycompany.myapp` to `src` as its package prefix and store the corresponding classes right in `src`.

Dependencies

Module dependencies are the entities that a module uses and, consequently, depends on. So, the term module dependency may refer to:

- The SDK associated with the module
- Libraries used in the module
- Other modules belonging to the same project

The term module dependency may also be used in a narrower sense meaning only the modules that a module depends on.

- Order of dependencies
- Dependencies on other modules

Order of dependencies

The module dependencies are processed in the order in which they appear in the corresponding list (see Dependencies tab).

At compile time, the order of items in the list defines the order in which the compiler (javac) looks for the classes to resolve the corresponding references. At runtime, this list defines the order in which the JVM searches for the classes to be loaded.

In certain situations, the order of libraries may be important. For example, if you want to use patches to certain libraries, these patches should be placed in the list before (above) the main libraries. Otherwise, new versions of the classes (contained in the patches) will be ignored.

For compilation, the position of an item in the list in relation to the module SDK (JDK) also matters, and in certain cases may be very important. (The module SDK is always shown in the list among the module dependencies.)

- All the items after the JDK are translated into the module `classpath` entries. Most of the libraries that you normally deal with should go there.
- All the items before the JDK correspond to the module `bootclasspath` entries.
  For the compilation to succeed, certain libraries (e.g. the ones having to do with the JDK itself and its extensions) must be included in the module `bootclasspath`. So, such libraries must appear in the list before the JDK.

  The use of such libraries, however, may be necessary in a very limited number of special cases. Thus, for the most of your modules, the JDK will (and should) be the first item in the list of dependencies.

Dependencies on other modules

If a module (module A) depends on another module (module B), IntelliJ IDEA assumes that:

- The sources of module A depend on the sources in module B and don't depend on its test sources.
- The test sources of module A depend on the sources and the test sources of module B.

Facet

Facets represent various frameworks, technologies and languages used in a module . They let IntelliJ IDEA know how to treat the module contents and thus ensure conformity with the corresponding frameworks and technologies.

Using facets enables you to download and configure the necessary framework components, have various descriptors generated automatically and stored in proper locations, and so on.

Most facets may be added to a module irrespective of whether other facets exist.

There are also facets that extend other facets. Such facets are possible only if a module has their parent facet. For more information, see Available Facets and Their Dependencies .

The list of available facets depends on which plugins are currently enabled in IntelliJ IDEA.

Available Facets and Their Dependencies

The following table lists the available facets . For each facet, it specifies a parent, and shows whether one or more facets are allowed for a module .

The information in this table may be read like this:

- A module can have only one Android facet. (It may have none, of course.)
- A module can have one or more EJB facets.
- A module can have one or more Google App Engine facets but only if it has Web facets. (A Google App Engine facet is an extension of a Web facet.) Only one Google App Engine facet is allowed per Web facet.

Facet	Parent	Max number	Notes
Android	Module	1	
AspectJ	Module	1	
Bean Validation	Module	1	
CDI	Module	> 1	CDI in the UI of IntelliJ IDEA is not shown as a facet. Instead, it is represented by the corresponding CDI implementation library in module dependencies. For more information, see Context and Dependency Injection (CDI) .
EJB	Module	> 1	
Google App Engine	Web facet	> 1 (1 per Web facet)	
GWT	Module	1	
Hibernate	Module	1	
JavaEE Application	Module	> 1	
JPA	Module	1	
JSF	Web facet	> 1 (1 per Web facet)	A JSF facet provides an additional ability of downloading the following JSF component libraries: ICEfaces, OpenFaces, PrimeFaces, RichFaces.
OSGi	Module	1	
Seam	Module	1	
Spring	Module	1	A Spring facet provides an additional ability of downloading the libraries for the following frameworks and Spring extensions: Spring Data JPA, Spring Integration, Spring Security, Spring Web Services, Spring Web Flow.
Struts	Web facet	> 1 (1 per Web facet)	
Struts 2	Web facet	> 1 (1 per Web facet)	
Tapestry	Module	1	
Web	Module	> 1	
Web Services	Web facet	> 1 (1 per Web facet)	
Web Services Client	Module	1	

Library

A library is a collection of compiled code used in your modules as is. A Java library, for example, can include class files, archives and directories with class files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`).

Libraries may optionally include the source code for the library classes as well as corresponding API documentation. Including the source files and documentation is optional and doesn't alter the way the library is used in any way. However, it adds the ability to use inline documentation extracted from the source code, and also to view the API documentation right from the IDE.

– Library levels
– Application server libraries
– Excluded library items

Library levels

You can define the libraries at the global (IDE), project, and module levels. The level of a library defines its usage scope:

– Global libraries can be used in any of your projects . That is, they can be added to dependencies of any module in any project.
– Project libraries can be used in any of the modules within the corresponding project. However, they are not available outside of the project in which they are defined.
– Module libraries exist only as dependencies of the corresponding module.

IntelliJ IDEA lets you move the libraries to a higher level (e.g. from project to global), create library copies at a lower level (e.g. you can create a copy of a global library at the project level), etc.

Application server libraries

Application server libraries let you use classes available in corresponding server distributions.

When you add a definition of an application server in IntelliJ IDEA, normally, an associated application server library is created.

In terms of the library levels, the application server libraries are global libraries. They can be added to dependencies of any module in any of your projects.

Excluded library items

You can make library items "excluded". IntelliJ IDEA will ignore such items when you write your code. As a result, the classes from excluded packages won't be present in code completion suggestion lists, references to such classes will be shown in the editor as unresolved, etc. However, when you compile or run your code, a library is still used as a whole, irrespective of whether there are excluded items in that library or not.

You can exclude folders, archives (e.g. JARs) and folders within the archives.

When your code references only a small portion of a big library, making the "unnecessary" items excluded may considerably increase the IDE performance.

In the UI, the excluded library items are shown as .

Path Variables

To avoid complications when moving projects from one computer to another, IntelliJ IDEA provides path variables. (A path variable, obviously, is a variable whose value is an absolute path to a directory or file.)

Path variables are particularly useful when working with third-party libraries stored outside the project directory.

To illustrate, let's assume there is a project shared among a team of developers (e.g., through version control), and there is a library in this project defined at the project or module level.

All is well if this library is located in the project directory or one of the module content roots . This, however, is rarely the case.

A more typical situation is when the library location is external to the project and modules. Such a library is referenced by its absolute path and there's no guarantee that this path is the same on every one of the computers used by the team.

The obvious solution is to define a path variable for the library location. In such a case, the library path may be set individually on each of the computers.

To summarize, you should define path variables under the following set of conditions:

- There are third-party libraries in your project defined at the project or module level.
- These libraries are stored outside the project directory and the module content roots.
- There is a need to work with the project on more than one computer (e.g., share the project among the development team members through version control).

Ignored variables

If when opening a project, IntelliJ IDEA detects unresolved path variables, it asks you to define proper values for them. If for some reason you don't want to do that (e.g., you are not going to use the corresponding library or libraries), you have an option of adding the corresponding variables to a list of ignored variables.

There may also be other cases when the list of ignored variables is useful.

At the internal level, path variables are represented by strings in which the name of a variable is enclosed between a pair of dollar sign characters, for example, `$MY_PATH_VARIABLE$` .

Such a pattern, in principle, may occur in your project without the meaning of a path variable. For example, `$SOME_STRING$` may occur within program parameters passed to the JVM in a run/debug configuration, etc.

To tell IntelliJ IDEA that a string that starts and ends with a dollar sign character (e.g., `$SOME_STRING$`), actually, is not a path variable, you should add such a string (e.g., `SOME_STRING`) to the list of ignored variables.

Path variables and the list of ignored variables are configured on the Path Variables page of the Settings dialog.

Supported Languages

In this section:

- Supported languages
- Coding assistance

Supported languages

Development of a modern application involves using multiple languages, that is why IntelliJ IDEA is an IDE for polyglot programming. With the deep understanding of all the subtleties of the source code structure and syntax, IntelliJ IDEA extends its support to:

- Java. IntelliJ IDEA supports Java up to version Java 8, with lambdas, type annotations etc. For the supported language level, IntelliJ IDEA provides code completion, code inspections, quick fixes and more.
- XML / XSL. Refer to the section Markup Languages and Style Sheets.
- Groovy. Refer to the section Groovy.

- JSP/JSPX
- Flex: advanced coding assistance for ActionScript, MXML, including code completion and inspections. Refer to ActionScript and Flex.
- HTML / XHTML. Refer to the section Markup Languages and Style Sheets.
- CSS3: coding assistance and compilation for Less; basic support of Sass 3.

  Refer to Markup Languages and Style Sheets.
- Stylus. Refer to section Transpiling Sass, Less, and SCSS to CSS.
- JavaScript. Refer to the section JavaScript-Specific Guidelines.
  - CoffeeScript. Refer to the section CoffeeScript Support.
  - TypeScript
- PHP up to version 5.4.0 with support of syntax highlighting in `.ini` files. Refer to the section PHP-Specific Guidelines.
- SQL and its dialects (SQL-92, MySQL, SQLite Oracle, PostgreSQL, and Derby). Refer to Databases and SQL.
- FreeMarker
- Velocity up to version 1.7
- IDL
- Spring-AOP
- Dart
- Drools Expert

Coding assistance

Coding assistance in IntelliJ IDEA includes:

- Syntax and error highlighting. The color attributes are configurable in the Colors and Fonts | <language> pages of the Settings dialog.
- File templates for the supported languages that enable creating stub classes, scripts etc.
- Live templates for creating complicated code constructs.
- Code completion.
- Code generation.
- Code folding, formatting, and highlighting.
- Intention actions and quick fixes.
- Ability to view code hierarchy.
- Quick access to the API documentation.
- Using macros in the editor.
- Advanced search and replace facilities.
- Advanced means of navigation.
- Refactoring.
- Import assistance.
- Language injection.
- The embedded local terminal where you can execute commands without leaving IntelliJ IDEA.

Besides editing assistance, IntelliJ IDEA enables debugging for Java, JavaScript, PHP, and Flex applications.

Warning! Debugging for JavaScript applications is supported only in the Chrome browser.

The polyglot arsenal of IntelliJ IDEA can be extended by installing plugins, for example, Scala, or La Clojure. With these plugins installed and enabled, IntelliJ IDEA provides the corresponding facets, coding assistance, running and debugging facilities.

Scope

On this page:

- Basics
- Types of scopes
- Defining scopes
- Scopes coloring
- Predefined scopes

Basics

A scope is a subset of files, packages and/or directories in your project, to which you can limit the application of specific operations, e.g. search , code inspection , insertion of copyright notices, etc. Besides, you can configure coloring for each scope to see at once what sort of file you are dealing with.

Scopes get more helpful as your project grows larger. There are a lot of predefined scopes that cover basic cases. Additionally, it is possible to add custom scopes to your project. For example, you can create custom scopes for tests or for the files you are responsible for in your team.

Types of scopes

Scopes can be either shared or local:

- Shared scopes are accessible for the team members via VCS and are stored at the project level. If the project has a file-based format , the shared scopes are stored in the `*.ipr` file; if the project has a directory-based format , the shared scopes are stored in the `scopes` directory under `.idea` , as a file with the extension `xml` , i.e. `.idea/scopes/<scope_name>.xml` .
- Local scopes are intended for personal use only and are stored in your workspace (`*.iws` file in the file-based project format, or in the file `workspace.xml` under `.idea` in the directory-based format) .

If necessary, you can share a local scope, make a shared scope local, or create a copy of the scope. For more information, see Configuring Scopes and File Colors .

Defining scopes

IntelliJ IDEA provides a special language that enables you to flexibly define the sets of entities included in a scope. See Scope Language Syntax Reference for details.

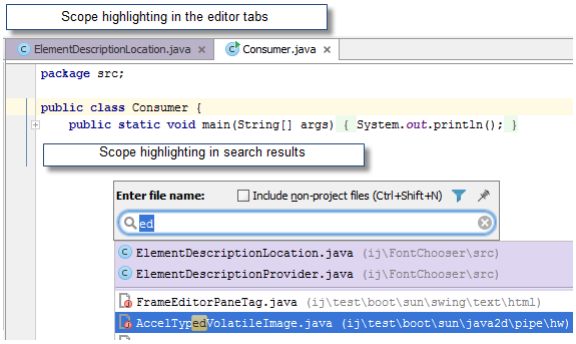
To create and edit scopes, use the Scopes page of the Settings dialog. Scopes are defined in the following modes:

- Manually, by specifying file masks according to the scope language syntax in the Pattern text box.
- By selecting files and folders and clicking the buttons Include, Include Recursively, Exclude, and Exclude Recursively. Based on the inclusion/exclusion, IntelliJ IDEA creates an expression and displays it in the Pattern.
  Refer to the section Configuring Scopes and File Colors .

You can explore the available scopes in the Project tool window .

Scopes coloring

Files belonging to different scopes can be highlighted in different colors throughout the IntelliJ IDEA's user interface: in navigation lists , in the editor tabs, in the Project window . This allows much faster and easier navigation in large projects.



If some file is included into several scopes, the order of the scopes becomes important: IntelliJ IDEA uses the color of the uppermost scope (shown in the Scopes settings page) to highlight such file. Of course, you can change the order of the scopes, and thus the resulted highlighting.

For detailed instructions on how to configure the scope order and scope-color associations, see Configuring Scopes and File Colors .

Predefined scopes

IntelliJ IDEA provides a number of predefined scopes, for example:

- Project Files. This scope includes all the files within the project content roots (see Module Contents). Module dependencies (libraries and SDKs), generally, are not included in this scope.
- Problems. This scope includes the files within the project content roots in which syntactic errors are found.
- Project and Libraries. This scope includes all the files within the project content roots, and also all module dependencies, libraries and SDKs.
  In the Project tool window , this scope corresponds to the scope view All.
- Project Production Files. This scope is similar to the Project Files scope . The difference is that the test source roots are not included. In the Project tool window , this scope corresponds to the scope view Production.
- Project Test Files. This scope is limited to the project test source roots. In the Project tool window , this scope corresponds to the scope view Tests.
- Non-Project Files. This scope is available only as a view in the Project tool window . It is limited to module dependencies (libraries and SDKs).
- Changed Files. This scope corresponds to all changed files, that is, ones associated with all existing changelists .
- Default. This scope corresponds to the files associated with the changelist `Default` .
- Favorite ' `<name>` '. This scope corresponds to a list of favorite items with the specified name. See Managing Your Project Favorites .
- Open files. This scope corresponds to the files opened in IntelliJ IDEA editor.

- Current file. This scope corresponds to the file currently active in IntelliJ IDEA editor.
- Selected files. This scope corresponds to the files currently selected in IntelliJ IDEA (e.g. in the Project tool window).

Predefined scopes cannot be edited.

Encoding

Different types of files use different ways to define encoding. IntelliJ IDEA recognizes encoding of files based on their contents.

Encoding has influence on the way IntelliJ IDEA reads or writes files. If a file has been modified but not yet saved, any changes in encoding affect file writing; if a file has not been modified, then reading is affected. IntelliJ IDEA suggests specific ways to change encoding of a file according to its type, using File Encodings Settings page, the Status bar, or the editor .

Note Encoding applies to directories and individual files. The encoding information saved in a file overrides the default encoding; encoding of a file or subdirectory overrides encoding settings on the higher levels.

Encoding	Can be changed in
File encoding is specified within the file, for example, in XML.	If a file contains explicit encoding declaration, you can change it in the Editor . In this case IntelliJ IDEA provides code completion.
File encoding is defined by BOM.	In this case, you can't change encoding with which IntelliJ IDEA reads the file, but it is still possible to change encoding for writing such file.
UTF characters are detected in the file contents.	IntelliJ IDEA provides an option that automatically changes file encoding to UTF , if the file contents can be reasonably interpreted as UTF. This option only works for reading; a file can be saved with any encoding.
Encoding cannot be found out from the file content.	In this case, the default encoding is the one defined by the IDE encoding in the File Encodings page of the Settings dialog. You can change it for multiple files and directories , or for a single file .
Properties files.	For the properties files, system default encoding is used. Though the characters in the other encodings are processed as unknown symbols, it is still possible to use them, by feeding such characters as escape sequences (for which IntelliJ IDEA provides the Transparent native-to-ASCII conversion option), or defining the default encoding for properties files.

Code Analysis

IntelliJ IDEA performs code analysis by applying inspections to your code. Numerous code inspections exist for Java and for the other supported languages.

The inspections detect not only compiling errors, but also different code inefficiencies. Whenever you have some unreachable code, unused code, non-localized string, unresolved method, memory leaks or even spelling problems – you'll find it very quickly.

IntelliJ IDEA's code analysis is flexibly configurable. You can enable/disable each code inspection and change its severity , create profiles with custom sets of inspections, apply inspections differently in different scopes , suppress inspections in specific pieces of code, and more.

The analysis can be performed in several ways:

- By default, IntelliJ IDEA analyses all open files and highlights all detected code issues right in the editor. On the right side of the editor you can see the analysis status of the whole file (the icon in the top-right corner).
  When an error is detected, this icon is ; in case of a warning, it is ; if everything is correct, the icon is .
- Alternatively, you can run code analysis in a bulk mode for the specified scope, which can be as large as the whole project.
- If necessary, you can apply a single code inspection in a specific scope.

For the majority of the detected code issues, IntelliJ IDEA provides quick fix suggestions . You can quickly review errors in a file by navigating from one highlighted line to another by pressing `F2` `Shift+F2` .

IntelliJ IDEA analyses your projects on the various levels:

- "On-the-fly" analysis helps you fix problems as they arise as you type, using intention actions .
- Code inspections are intended to point out issues related to the program design.
- Highlighting level allows you to control the scope of problems highlighted in the current file.
- Dependencies analysis helps you understand the structure of your source code, explore relationships between the components of your projects, track down dependencies and work through the code hierarchies.
- Dataflow analysis helps you with code archeology.
- Reverse engineering .

New Project Wizard

To access the New Project wizard:

If no project is currently open: select Create New Project option on the Welcome screen

Otherwise: File | New | Project

Use the New Project wizard to create a new project from scratch.

- Project Category and Options
- Project Template
- J2ME Page
- Gradle Page
- Maven Page
- New Project Wizard Android Dialogs
- Create New Project: HTML5 Boilerplate
- Create New Project: Web Starter Kit
- Create New Project: React Starter Kit
- Create New Project: Twitter Bootstrap
- Create New Project: Foundation
- Create New Project: Node.js Express App
- Create New Project: Meteor Application
- Create New Project: PhoneGap/Cordova
- Create New Project: Yeoman
- Create New Project: Composer Project
- Create New Project: Drupal Module
- Create New Project: Google App Engine for PHP
- Create New Project: PHP Empty Project
- Project Name and Location

New Wizard Scala Plugin Pages

IntelliJ IDEA lets you use this page for selecting an additional framework for your new Scala project.

Item	Description
Scala	Select this option to create a project with Scala SDK.
SBT	Select this option to create an SBT-based project.
Activator	Select this option to create a Scala project with the Typesafe Activator templates.
Play2	Select this option to create a Scala project with the Play2 framework.

Typesafe Activator New Project Dialog

IntelliJ IDEA lets you use this page for specifying the Typesafe Activator project information.

Item	Description
Project Name	Use this field to specify the name of your project.
Project Location	Use this field to specify the location of your project. You can click … and select the necessary directory in the dialog that opens .
Project SDK	Specify the SDK to be used. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select the installation folder of the desired IntelliJ IDEA version. (An IntelliJ IDEA installation acts as an IntelliJ Platform Plugin SDK.) If asked to specify a JDK first, do so.
List of templates	Use this area to select an activator template for your project. The description of each template is available in the Description field.

Item	Description
More Settings	Click the arrow (▸ or ▾) to show or hide additional settings. Mainly, these are the settings for the module to be created (discussed below). Note that in certain cases those additional settings are unavailable.
Module name	Specify the module name.
Content root	Specify the path to the module content root folder. (This is where all the files that make up you module will be stored; for more information, see Content Root .) To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Module file location	Specify the path to the folder where the `.iml` module file should be created. By default, this file is created in the module content root folder (recommended). To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Project format	Select the project format to be used. (The `.idea` directory-based format is recommended).

Play2 New Project Dialog

IntelliJ IDEA lets you use this page for specifying the Play2 project information.

Item	Description
Project Name	Use this field to specify the name of your project.
Project Location	Use this field to specify the location of your project. You can click … and select the necessary directory in the dialog that opens .
Project SDK	Specify the SDK to be used. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select the installation folder of the desired IntelliJ IDEA version. (An IntelliJ IDEA installation acts as an IntelliJ Platform Plugin SDK.) If asked to specify a JDK first, do so.
Scala version	Use this field to specify the Scala SDK version. If the necessary SDK is already defined in IntelliJ IDEA, select it from the drop-down list.
Play 2 version	Use this field to specify the Play 2 SDK version. Note Please note that Play 2 versions 2.4 and later require Java 8.

Item	Description
More Settings	Click the arrow (▸ or ▾) to show or hide additional settings. Mainly, these are the settings for the module to be created (discussed below). Note that in certain cases those additional settings are unavailable.
Module name	Specify the module name.
Content root	Specify the path to the module content root folder. (This is where all the files that make up you module will be stored; for more information, see Content Root .) To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Module file location	Specify the path to the folder where the `.iml` module file should be created. By default, this file is created in the module content root folder (recommended). To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Project format	Select the project format to be used. (The `.idea` directory-based format is recommended).

Scala New Project Dialog

IntelliJ IDEA

Item	Description
Project Name	Use this field to specify the name of your project.
Project Location	Use this field to specify the location of your project. You can click … and select the necessary directory in the dialog that opens .
Project SDK	Specify the SDK to be used. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select the installation folder of the desired IntelliJ IDEA version. (An IntelliJ IDEA installation acts as an IntelliJ Platform Plugin SDK.) If asked to specify a JDK first, do so.
Scala SDK	Use this field to specify the Scala SDK. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click Create and select the SDK version you need to download.

Item	Description
More Settings	Click the arrow (▸ or ▾) to show or hide additional settings. Mainly, these are the settings for the module to be created (discussed below). Note that in certain cases those additional settings are unavailable.
Module name	Specify the module name.
Content root	Specify the path to the module content root folder. (This is where all the files that make up you module will be stored; for more information, see Content Root .) To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Module file location	Specify the path to the folder where the `.iml` module file should be created. By default, this file is created in the module content root folder (recommended). To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Project format	Select the project format to be used. (The `.idea` directory-based format is recommended).

SBT New Project Dialog

IntelliJ IDEA lets you specify the SBT project information.

Item	Description
Project Name	Use this field to specify the name of your project.
Project location	Use this field to specify the location of your project. You can click … and select the necessary directory in the dialog that opens .
Project SDK	Specify an SDK for your project. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and, in the dialog that opens , select the installation folder of the desired Java ME SDK. (By this time, the corresponding SDK must already be installed on your computer.)
SBT version	Use this drop-down list to specify the SBT version for your project.
Scala version	Use this drop-down list to specify the Scala version to be used in your project.
Use auto-import	Select this check box to resolve all the changes made to the SBT project automatically every time you refresh your project.
Create directories for empty content roots automatically	Select this check box to add the *src* directory to your project.
Download	Use this area to download the following options: – Sources - select this check box to download sources and docs for project dependencies. – Javadocs - select this check box to download Javadoc . – Sources for SBT and plugins - select this check box to download sources docs for SBT itself.

Item	Description
More Settings	Click the arrow (▸ or ▾) to show or hide additional settings. Mainly, these are the settings for the module to be created (discussed below). Note that in certain cases those additional settings are unavailable.
Module name	Specify the module name.
Content root	Specify the path to the module content root folder. (This is where all the files that make up you module will be stored; for more information, see Content Root .) To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Module file location	Specify the path to the folder where the `.iml` module file should be created. By default, this file is created in the module content root folder (recommended). To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Project format	Select the project format to be used. (The `.idea` directory-based format is recommended).

Project Category and Options

This page of the New Project wizard opens when you select File | New | Project in the main menu or Create New Project on the Welcome screen.

In the left-hand pane, select the project category. This may be the technology that you are going to use, the platform or runtime that your development is going to target, etc.

In the right-hand part of the page, select additional options and specify associated settings.

Don't worry about selecting "wrong" options at the moment. Just select the ones that you think suit you best. If necessary, you will be able to make the necessary changes to your project at a later time.

Note that the set of options you can select from depends on which plugins are currently enabled in IntelliJ IDEA.

- Java
- Java Enterprise
- J2ME
- Android
- Clouds
- Spring
- Java FX
- IntelliJ Platform Plugin
- Spring Initializr
- Maven
- Gradle
- Groovy
- Grails
- Griffon
- PHP
- Kotlin
- Static Web
- Flash
- Empty Project

Java

Select this option if you are going to develop a Java application.

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)

If necessary, select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

Java Enterprise

Select this option if you are going to develop a Java EE application.

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Java EE version	Select the Java EE version to be supported. (Affects the corresponding version setting for the Web Application, EJB and JavaEE Application options.)
Application Server	Specify the application server that you are going to use to deploy and run your application. As a result, IntelliJ IDEA will create a run/debug configuration for the specified server. (You can specify the server later.) You can select a server which IntelliJ IDEA is already aware of, or specify another "new" server. To specify a new server, click New and select the server of interest. Then, specify the server settings: – For a server installed locally, specify the path to the server installation directory. (Click … to select the directory in the corresponding dialog .) – For a hosted server (Cloud Foundry or CloudBees), specify your user account details.

Select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

J2ME

Select this option if you are going to develop for Java ME .

Item	Description
Project SDK	Specify an SDK for your project. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and, in the dialog that opens , select the installation folder of the desired Java ME SDK. (By this time, the corresponding SDK must already be installed on your computer.)

SQL Support	Select the check box to enable SQL support. Select the SQL dialect to be used by default from the list.

Android

Select this option if you are going to develop for the Android OS. Select:

- Gradle: Android Module to develop a Gradle-based Android application or library.
- Application Module to develop an Android application.
- Empty Module to develop an Android application or library.
- Library Module to develop a shared Android library.

Clouds

Select this option if you are going to deploy your application to a cloud platform such as CloudBees, Cloud Foundry, Heroku or OpenShift. See also, Working with Cloud Platforms.

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens, select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Account	Specify your cloud user account. If the corresponding user account is already registerd in IntelliJ IDEA, select it from the list. Otherwise, click New, select the cloud platfrom and specify your user account settings in the dialog that opens.

Application	Cloud platform-specific application settings.
	CloudBees and Cloud Foundry. IntelliJ IDEA will create a sample Java web application which you'll be able to deploy to the cloud and run straight away.
	– Version. The version of the Servlet specification to be supported.
	– Create web.xml. For version 3.0 or later: select this check box to create the deployment descriptor file `web.xml`. (For earlier versions, this file is always created.)
	
	Heroku. You can select to create a new application or to git-clone the source code for one of your applications already deployed on Heroku.
	– Template. A new sample application will be created. You'll be able to deploy this application to Heroku straight away.
	– Existing. Select the application whose source code you want to clone.
	
	OpenShift. You can select to git-clone the source code for one of your applications already deployed OpenShift or to create a new application.
	– Existing. Select the application whose source code you want to clone.
	– New. Select this option to create a new application. Specify the settings for your new application (for OpenShift terminology, see OpenShift documentation):
	  – Standalone Cartridge. Select the primary cartridge.
	  – Gear size. Select the gear size.
	  – Scaling. Select the check box if the application should be scalable.
	  – Embeddable Cartridges. Select (additional) embedded cartridges.

Spring

Select this option if you are going to develop a Spring application.

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens, select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)

For information on other options and settings, see:

- Spring
- Spring MVC, Spring Batch, or other Spring framework
- Additional Libraries and Frameworks

See also, Enabling Spring Support.

Java FX

Select this option if you are going to develop a JavaFX application.

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK (version 7 or later) is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens, select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)

IntelliJ Platform Plugin

Select this option if you are going to develop a plugin for IntelliJ IDEA or other IntelliJ Platform-based IDE.

Item	Description

Project SDK	Specify an SDK for your project. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and, in the dialog that opens , select the installation folder of the desired IntelliJ IDEA version. (An IntelliJ IDEA installation acts as an IntelliJ Platform Plugin SDK.) (By this time, the corresponding IntelliJ IDEA version must already be installed on your computer.)
Groovy	Select the check box to be able to use Groovy . Specify the Groovy installation to be used. Use library. If the desired version of Groovy is already defined in IntelliJ IDEA, select it from the list. (Groovy in IntelliJ IDEA is represented by a library .) Create. Click this button to create a library for Groovy. In the dialog that opens , select the Groovy installation directory.
SQL Support	Select the check box to enable SQL support. Select the SQL dialect to be used by default from the list.

Spring Initializr

Select this option if you are going to develop a Spring Boot application.

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Initializr Service URL	Specify the Spring Initializr instance URL. By default, it is `https://start.spring.io` , but you can use any other custom instance if needed.

Maven

Select this option if you are going to develop a Java application with dependencies managed by Maven .

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Create from archetype	If this check box is not selected, the new `pom.xml` file will contain the basic information. If this check box is selected, the new module will be created on the base of a Maven archetype chosen from the list that includes both the standard archetypes, and the ones found in Maven indices. You can modify Maven properties on Maven Settings Page . If you want to populate the list with some archetype from a remote Maven repository, click the Add Archetype button, and find the desired archetype by Maven coordinates specified in Add Archetype Dialog .

Gradle

Select this option if you are going to develop a Java application with dependencies managed by Gradle .

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Use auto-import	Select this check box to resolve all the changes made to the Gradle project automatically every time you refresh your project.
Use default gradle wrapper (recommended)	Select this option to use Gradle wrapper. Using Gradle wrapper lets you get automatic Gradle download for the build. It also lets you build with the precise Gradle version.
Use customizable gradle wrapper	Select this option to use Gradle wrapper customization in script.
Use local gradle distribution	Select this option to run local build scripts.
Gradle home	Use this field to specify the fully qualified path to your Gradle installation. This field becomes active when you select Use local gradle distribution.

Groovy

Select this option if you are going to develop a Groovy application.

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Groovy library	If the desired version of Groovy is already defined in IntelliJ IDEA, select it from the list. (Groovy in IntelliJ IDEA is represented by a library .) Create. Click this button to create a library for Groovy. In the dialog that opens , select the Groovy installation directory.

If necessary, select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

Grails

Select this option if you are going to develop a Grails application.

See also Creating a Grails Application Module .

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Grails library	If the desired version of Grails is already defined in IntelliJ IDEA, select it from the list. (Grails in IntelliJ IDEA is represented by a library .) Create. Click this button to create a library for Grails. In the dialog that opens , select the Grails installation directory.

If necessary, select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

Griffon

Select this option if you are going to develop a Griffon application.

See also Creating a Griffon Application Module .

Item	Description
Project SDK	Specify an SDK (JDK) for your project. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Griffon library	If the desired version of Griffon is already defined in IntelliJ IDEA, select it from the list. (Griffon in IntelliJ IDEA is represented by a library .) Create. Click this button to create a library for Griffon. In the dialog that opens , select the Griffon installation directory.

If necessary, select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

PHP

Select PHP if you are going to develop an application using PHP . This option is available only if the PHP plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

In the right-hand pane, choose one of the following project types:

- PHP Empty Project: choose this option to get just a project folder without any contents. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: PHP Empty Project for a detailed description of the fields.
- Composer Project: choose this option to have a project stub created using the Composer template . Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: Composer Project for a detailed description of the fields.
- Drupal Module: choose this option to have a Drupal module created and set up. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: Drupal Module for a detailed description of the fields. The option is available only if the Drupal Support plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
- WordPress Plugin: choose this option to get a project stub for developing applications using WordPress, see Using WordPress Content Management System for details. In the right-hand pane, type the project location and specify the folder where WordPress is installed. This folder should contain the `wp-admin` and `wp-includes` subdirectories.

  The option is available only if the WordPress Support plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
- App Engine Project: choose this option to get a project stub for developing applications that will run in the Google PHP Runtime Environment , see Using Google App Engine for PHP for details. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see for a detailed description of the fields.
  The option is available only if the Google App Engine for PHP plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Kotlin

Select this option if you are going to create a Kotlin project. Specify the associated settings.

Item	Description
Project name	Specify the project name.
Project location	Specify the path to the directory in which you want to create the project. (By default, a directory having the same name as the project is created.) You can click ... (Shift+Enter) and select the necessary directory in the dialog that opens . (You can create a new directory in that dialog, e.g. by using .)
Project SDK	Specify an SDK for your project. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select SDK type. Then, inkotlin_intro_p dialog that opens, select the installation folder of the desired SDK. (By this time, the corresponding SDK must already be installed on your computer. If it isn't, download and install it first.)
Kotlin runtime	Specify here the runtime library `kotlin-runtime.jar` . The library resides within the Kotlin plugin and contains the standard Kotlin classes. If the desired library is missing, click Create. When a project is being created, one can either copy the said jar to the project (option Copy to), or just refer to the jar from the Kotlin plugin (option Use library from plugin).

More Settings	Click the arrow (or) to show or hide additional settings. Mainly, these are the settings for the module to be created (discussed below). Note that in certain cases those additional settings are unavailable.
Module name	Specify the module name.
Content root	Specify the path to the module content root folder. (This is where all the files that make up you module will be stored; for more information, see Content Root .) To use a different folder, click (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Module file location	Specify the path to the folder where the `.iml` module file should be created. By default, this file is created in the module content root folder (recommended). To use a different folder, click (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Project format	Select the project format to be used. (The `.idea` directory-based format is recommended).

Static Web

Select Static Web if you are going to develop a Web application using HTML / CSS , JavaScript , Node.js , and related frameworks.

Choose this option also if you want to generate a project stub based on a framework template.

In the right-hand pane, choose one of the following project types:

- Static Web: choose this option to get just a project folder without any contents.
- HTML5 Boilerplate: choose this option to have the project structure set up and some sources generated based on the HTML5 Boilerplate template. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: HTML5 Boilerplate for a detailed description of the fields.
- Web Starter Kit: choose this option to have the project structure set up and some sources generated in accordance with the Web Starter Kit requirements. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: Web Starter Kit for a detailed description of the fields.
- React Starter Kit: choose this option to have the project structure set up and some sources generated according to the React requirements. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: React Starter Kit for a detailed description of the fields.
- Twitter Bootstrap: choose this option to have the project structure set up and some sources generated based on the Twitter Bootstrap template. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: Twitter Bootstrap for a detailed description of the fields.
- Foundation: choose this option to have the project structure set up and some sources generated based on the Foundation framework template. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: Foundation for a detailed description of the fields.
- AngularJS: choose this option to have the project structure set up and some sources generated based on the AngularJS framework template.
- Dart Web Application: choose this option to have the project structure set up and some sources generated for a Dart application.
- Node.js Express App: choose this option to have the project structure set up and some project sources generated based on the Express framework. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: Node.js Express App for a detailed description of the fields.

  The option is available only if the Node.js plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
- PhoneGap/Cordova App: choose this option to have the project structure set up and some sources generated based on the PhoneGap , Apache Cordova , and Ionic frameworks. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: PhoneGap/Cordova for a detailed description of the fields.
- Meteor App: choose this option to have the project structure set up and some sources generated based on the Meteor frameworks. Click Next, and then specify the project name, location, and other settings in the dialog box that opens, see Create New Project: Meteor Application for a detailed description of the fields.

Flash

Select this option if you are going to develop for the Adobe Flash runtimes using Flex or ActionScript . Specify the associated settings.

Item	Description
Target platform	Select the target environment for the content that you are going to develop: – Web for Flash player / Web browser-targeted content. – Desktop for Adobe AIR-targeted content. – Mobile for the content intended for mobile devices (Android, iOS, etc.).
Pure ActionScript	Select this check box if you are not going to use MXML (i.e. all your source code will be written in ActionScript).
Output type	Select the intended output type, that is, what your resulting content is going to be: – Application. A runnable application, an SWF file. – Runtime-loaded module. A dynamically-loadable module , an SWF file. – Library. An SWC file.
Target devices	For a Mobile Application: use the Android and iOS check boxes to specify the intended target devices for your application. As a result, IntelliJ IDEA enables or disables creating an application descriptor and packaging your application for the corresponding devices. (The Android and iOS check boxes on this page correspond to the Enabled check boxes on the Android and iOS tabs in the build configuration that will be created.)
Flex/AIR SDK	Select the Flex or AIR SDK to be used. If the list is empty or does not contain the required SDK, click (Shift+Enter) and add the required SDK in the Configure SDK dialog.
Target player	For the Web target platform: the target Flash Player version (readonly). (This setting is defined by the selected Flex SDK version.)
Create sample app	For the Application output type: select this check box if you want a sample application to be created. You can use this sample application for learning and also as a basis for your own application development.

	If necessary, change the source file name suggested by IntelliJ IDEA.
Create HTML wrapper template	For a Web Application: select this check box if you want an HTML wrapper template for your application to be created. Select or deselect the associated options as needed: – Enable integration with browser navigation . Select this option to enable deep linking. Deep linking lets users navigate their interactions with the application by using the Back and Forward buttons in their browser. – Check Flash player version . If you select this option, the compiled application will check for the correct version of Flash Player. – Express install . If you select this option, the application will run an SWF file in the existing Flash Player to upgrade users to the latest version of the player.

Empty Project

If you select this option, IntelliJ IDEA will create just a minimal folder structure and the necessary project definition files. You'll be able to expand your project later.

Additional Libraries and Frameworks

Select the technologies, frameworks and languages to be supported. If necessary, also select an application server that you want to use to deploy and run your application.

As a result, you'll be able to download and configure the necessary framework components (e.g. implementation libraries). You will also have various descriptors, configuration files and sample code generated automatically, and the module dependencies properly configured.

In IntelliJ IDEA terms, the necessary facets will be created along with an artifact specification and a run/debug configuration , if appropriate.

Note that the set of frameworks and technologies you can select from depends on which plugins are currently enabled in IntelliJ IDEA. For example, Ruby and Python are available only if the corresponding plugins are installed and enabled. For more information, see Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

- Web Application
- Struts
- Struts 2
- WebServices
- JSF
- Primefaces, Richfaces, Openfaces, or Icefaces
- Google App Engine
- Groovy
- Hibernate
- JavaEE Persistence
- JBoss Drools
- OSGi
- SQL Support
- Thymeleaf
- WebServices Client
- Batch Applications
- CDI: Contexts and Dependency Injection
- DM Server
- EJB: Enterprise JavaBeans
- Google Web Toolkit
- JavaEE Application
- RESTful WebServices
- Tapestry
- Spring
- Spring MVC, Spring Batch, or other Spring framework
- Application Server

Web Application

Select the check box to enable generic Web application development support. See also, Enabling Web Application Support .

Item	Description
Version	Select the version of the Servlet specification to be supported.
Create web.xml	For version 3.0 or later: select this check box to create the deployment descriptor file `web.xml` . (For earlier versions, this file is always created.)

Struts

Select the check box to enable Apache Struts 1.x support. See also, Preparing to Use Struts .

Item	Description
Version	Select the Struts version to be supported. If you also choose to download the library files that implement Struts (the Download option), the selected version will define which files you will be able to download.
Libraries	You'll need a library that implements Struts. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement the selected Struts version. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Struts 2

Select the check box to enable Apache Struts 2 support. See also, Preparing to Use Struts 2 .

You'll need a library that implements Struts 2. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the library files that implement Struts 2. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
- Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

WebServices

Select the check box to enable Web Services development support. See also, Preparing to Develop a Web Service .

Item	Description
Generate sample server code	Select this check box to have a sample `HelloWorld` class created in your source folder (e.g. `src`).
Configure	Click this link to specify the settings for WS engines that you want to use. (The Web Services dialog will open.)

JSF

Select the check box to enable JavaServer Faces (JSF) support. See also, Preparing for JSF Application Development .

Item	Description
Version	Select the JSF version to be supported.
Libraries	You'll need a library that implements JSF. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement JSF. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Primefaces, Richfaces, Openfaces, or Icefaces

Select the check box to be able to use the corresponding JSF component library (PrimeFaces , RichFaces , OpenFaces , or ICEfaces). See also, Preparing for JSF Application Development .

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the corresponding JSF component library files. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Google App Engine

Select the check box to be able to use Google App Engine . See also, Creating Google App Engine Project .

Item	Description
Google App Engine SDK	Specify the path to the Google App Engine SDK for Java installation directory. You can click … and select the corresponding directory in the dialog that opens .
Persistence	If necessary, select the App Engine Datastore implementation to be supported (JDO or JPA).
Download	If the path to Google App Engine SDK is not specified, you can click this link to open the Google App Engine Downloads page . (This page lets you download the latest version of Google App Engine SDK for Java.)

Groovy

Select the check box to enable Groovy support. See also, Creating Modules with Groovy and Groovy-Based Frameworks Support .

Select an existing Groovy library or create a new library for Groovy:

- Use library. Select the Groovy library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
- Create. If Groovy is already installed on your computer, you can create a library for Groovy and use that new library. To do that, click Create and select the Groovy

installation directory in the dialog that opens .

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

Hibernate

Select the check box to enable Hibernate support. See also, Enabling Hibernate Support .

Item	Description
Create default hibernate configuration and main class	Select this check box to have a Hibernate configuration file `hibernate.cfg.xml` and a class with the `main()` method created.
Import database schema	Select this check box to have a database schema imported automatically.
Libraries	You'll need a library that implements Hibernate. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement Hibernate. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

JavaEE Persistence

Select the check box to enable Java Persistence API (JPA) support. See also, Enabling JPA Support .

Item	Description
persistence.xml version	Select the version of the `persistence.xml` file that you want to use. If you also choose to download the library files that implement JPA (the Download option), the selected version will define which files you will be able to download.
Import database schema	Select this check box to have a database schema imported automatically. Optionally, select the JPA implementation-specific database scheme to be imported from the list above the check box.
Libraries	You'll need a library that implements JPA. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement the selected JPA version. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

JBoss Drools

Select the check box to enable JBoss Drools support.

You'll need a library that implements Drools. You can choose to use an existing library, create and use a new one, or download the library files if they are not yet available on your computer.

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

– Download. Select this option to download the library files that implement Drools. (The downloaded files will be arranged in a library .)
Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

OSGi

Select the check box to enable OSGi support.

You'll need a library that implements OSGi. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click

Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

- Download. Select this option to download the library files that implement OSGi. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
- Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

SQL Support

Select the check box to enable SQL support.

Item	Description
Default Dialect	Select the SQL dialect to be used by default for the module. Select Project Default to use the default project SQL dialect.

Thymeleaf

Select the check box to enable Thymeleaf support.

You'll need a library that implements Thymeleaf. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the library files that implement Thymeleaf. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
- Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

WebServices Client

Select the check box to enable Web Services client development support. See also, Enabling Web Service Client Development Support .

Item	Description
Generate sample client code	Select this check box to have sample client code generated in your source folder (e.g. `src`). To generate the code, IntelliJ IDEA will ask you to specify the corresponding WSDL file.
Configure	Click this link to specify the settings for WS engines that you want to use. (The Web Services dialog will open.)

Batch Applications

Select the check box to enable Batch Applications development support.

Item	Description
Create batch.xml	Select the check box to create a `META-INF\batch.xml` mappings file (one with the `<batch-artifacts>` root element).
Create Sample Job Xml	Select the check box to create a sample job XML file (`META-INF\batch-jobs\job.xml`).
Libraries	You'll need a library that implements the batch framework. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the files that implement the batch framework. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

CDI: Contexts and Dependency Injection

Select the check box to enable Contexts and Dependency Injection (CDI) support.

You'll need a library that implements CDI. You can choose to use an existing library, create and use a new one, or download the library files if they are not yet available on your computer.

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click

Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

– Download. Select this option to download the library files that implement CDI. (The downloaded files will be arranged in a library .)
Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

DM Server

Select the check box to be able to use SpringSource dm Server and develop dm Server-targeted applications.

Item	Description
Server	Select the server to be used from the list (if the corresponding server is already defined in IntelliJ IDEA). To define a server in IntelliJ IDEA, click Add and specify the server settings in the Spring dmServer dialog that opens.
Facet	Select the deployment artifact type that the module will implement (the Spring DM facet type in IntelliJ IDEA terms): – Bundle, an OSGi bundle. IntelliJ IDEA will create: – The `META-INF\MANIFEST.MF` file for the bundle. – A dm Bundle artifact configuration . – A project-level library for the dm Server API. This library will be added to the module dependencies . – A dm Server-oriented run/debug configuration . If necessary, specify additional options . – PAR or Plan, a dm Server PAR or plan. Specify the associated settings . – Configuration, a configuration artifact. Specify the artifact name . For more information on dm Server deployment artifacts, see "Deployment Architecture" and "Developing Applications" in SpringSource dm Server Programmer Guide .
Bundle options	
Spring DM Support	Select this check box to enable Spring support (to create a Spring facet in IntelliJ IDEA terms). As a result, IntelliJ IDEA will create the following files: – `META-INF\spring\module-context.xml` – `META-INF\spring\osgi-context.xml` At this step you are not suggested to download the library files that implement Spring. However, you will be able to do that after the module has been created by using the corresponding quick fix in the Project Structure dialog. For more information, see Adding a Spring facet to an existing module.
Web module	Select this check box to enable generic Web application development support (to create a Web facet in IntelliJ IDEA terms). Specify the associated settings: – Version. Select the version of the Servlet specification to be supported. – Create web.xml. For version 3.0 or later: select this check box to create the deployment descriptor file `web.xml` . (For earlier versions, this file is always created.) As a result, IntelliJ IDEA will create `web\WEB-INF\web.xml` (for version 3.0 or later - if so specified).
PAR or Plan options	
Name	For a PAR, this is the application identifier (`Application-SymbolicName`), for a plan - the plan name (the `name` attribute of the `<plan>` element).
Version	The application or the plan version (`Application-Version` or the `version` attribute of the `<plan>` element).
Plan	Select this option to create a plan. IntelliJ IDEA will create: – A `.plan` XML file. – A dm Plan artifact specification.
Platform Archive (PAR)	Select this option to create a PAR. IntelliJ IDEA will create: – The `META-INF\MANIFEST.MF` file for the PAR. – A dm Platform Archive artifact specification.
Scoped	For a plan: select this check box to make the plan scoped (corresponds to `scoped="true"` within the `<plan>` element).
Atomic	For a plan: select this check box to make the plan atomic (corresponds to `atomic="true"` within the `<plan>` element).
Nested bundles	Use the controls in this area to manage other dm Server deployment artifacts within the PAR or plan. (In IntelliJ IDEA, these are represented by other modules within the same project if those modules have suitable dm Server facets. A PAR may include OSGi bundles and configuration artifacts; a plan - OSGi bundles, configuration artifacts, PARs and other plans). – Add. Use this button to add the artifacts to the list. Select the necessary artifacts (IntelliJ IDEA modules) in the dialog that opens. – Remove. Use this button to remove the selected artifacts from the list. – Up. For a plan: use this button to move the selected artifact one line up in the list. (The order of artifacts defines their deployment order.) – Down. For a plan: use this button to move the selected artifact one line down in the list. – Versions. For a plan: use this button to specify the version or the range of versions for the selected artifact (corresponds to the `version` attribute of the `<artifact>` element).
Configuration option	
Name	Specify the OSGi name of the artifact (at the deployment stage, corresponds to the name of the file).

EJB: Enterprise JavaBeans

Select the check box to enable Enterprise JavaBeans (EJB) support. See also, Enabling EJB Support .

Item	Description
Version	Select the EJB version to be supported. If you also choose to download the library files that implement EJB (the Download option), the selected version will define which files you will be able to download.

Libraries	You'll need a library that implements EJB. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement the selected EJB version. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Google Web Toolkit

Select the check box to be able to use Google Web Toolkit (GWT). See also, Enabling GWT Support .

Item	Description
GWT SDK	Specify the path to the GWT SDK installation directory. You can click ... and select the corresponding directory in the dialog that opens .
Create sample application	Select this check box to have a sample application created. Specify the package for the application classes in the field underneath.
Download GWT	If the path to GWT SDK is not specified: you can click this link to open the Google Web Toolkit Downloads page . (This page lets you download a GWT SDK.)

JavaEE Application

Select the check box to enable Java EE application development support. See also, Enabling Java EE Application Support .

Item	Description
Version	The Java EE version.

RESTful WebServices

Select the check box to enable RESTful Web Services (client and server) development support. See also, RESTful WebServices .

Item	Description
Generate server code	Select this check box to have a sample `HelloWorld` server class created in your source folder (e.g. `src`).
Generate client code	Select this check box to have a sample `HelloWorldClient` class created in your source folder (e.g. `src`).
Libraries	You'll need a library that implements the JAX-RS API. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the files that implement the JAX-RS API. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Tapestry

Select the check box to enable Apache Tapestry support. See also, Enabling Tapestry Support .

You'll need a library that implements Tapestry. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

– Download. Select this option to download the library files that implement Tapestry. (The downloaded files will be arranged in a library .)
Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

– Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Spring

Select the check box to enable Spring support. See also, Enabling Spring Support .

You'll need a library that implements Spring. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the library files that implement Spring. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
- Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Spring MVC, Spring Batch, or other Spring framework

Select a check box to add support for a particular Spring framework (e.g. Spring MVC , Spring Batch , etc.). See also, Enabling Spring Support .

You'll need a library that implements the selected framework. You can choose to use an existing library, create and use a new one, or download the library files if they are not yet available on your computer.

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the library files that implement the selected Spring framework. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Application Server

Select the check box and specify the application server that you are going to use to deploy and run your application. As a result, IntelliJ IDEA will create a run/debug configuration for the specified server.

Item	Description
Application Server	Select a server which IntelliJ IDEA is already aware of, or specify another "new" server. To specify a new server, select New <Server Name> from the list. Then, in the field or fields that appear underneath, specify the server settings: – For a server installed locally, specify the path to the server installation directory in the <Server Name> Home field. (Click … to select the directory in the corresponding dialog .) – For a hosted server (Cloud Foundry or CloudBees), specify your user account details.

Project Template

There are sample projects for various application types (e.g. Java, Java EE, Spring). Such projects are created according to pre-defined project templates and normally include sample code, a run configuration for that code, etc.

To create a template-based project, select the Create project from template check box and an option of interest in the area underneath.

J2ME Page

This page is available only for J2ME modules and provides JAD/JAM file settings depending on the selected J2ME SDK (WTK or DoJa). You can change these settings later in the Project Structure dialog (Project Structure | Module | Mobile Module Settings).

In this section:

– WTK

– DoJa

WTK

Item	Description
MIDlet-Name	In this text box, specify the MIDlet suite name (corresponds to the JAD MIDlet-Name property). This option is necessary to identify MIDlet suite on a device.
MIDlet-JAR-URL	In this text box, specify the MIDlet JAR location. This option is also necessary to identify MIDlet suite on a device. The JAR file will be installed from the location MIDlet-JAR-URL afterward. You can specify the location manually, or click the ellipsis button and select the necessary location in the dialog that opens .
MIDlet-Vendor	In this text box, specify the MIDlet vendor name, that is, the MIDlet suite provider.
MIDlet-Version	In this text box, specify your MIDlet version number.
Keep user-defined JAD file	Select this check box to preserve the `JAD` file (with settings specified above) to the project.

DoJa

Item	Description
AppName	In this text box, type the application name (50 bytes maximum).
PackageUrl	In this text box, specify the URL address to access the application. Make sure the URL is an ASCII-format string. Also, an IP address cannot be specified directly.
Keep user defined JAM file	Select this check box to add the `JAM` file containing settings specified above to the project.

Gradle Page

Use this page to specify Gradle settings for your project.

Item	Description
Use auto-import	Select this check box to resolve all the changes made to the Gradle project automatically every time you refresh your project.
Create directories for empty content roots automatically	Select this check box to create the default directory structure for a Gradle project, e.g. `/src/main/java` and `/src/test/java`.
Create separate module per source set	Select this check box to use the source set feature in resolving your Gradle projects.
Use default gradle wrapper (recommended)	Select this option to use Gradle wrapper. Using Gradle wrapper lets you get automatic Gradle download for the build. It also lets you build with the precise Gradle version.
Use gradle wrapper task configuration	Select this option to use Gradle wrapper customization in script.
Use local gradle distribution	Select this option to run local build scripts.
Gradle home	Use this field to specify the fully qualified path to your Gradle installation. This field becomes active when you select Use local gradle distribution. If Gradle location has been defined by the environment variables `GRADLE_HOME` or `PATH`, then IntelliJ IDEA deduces this location, and suggests this path as the default value. If Gradle location has not been deduced from the environment variables, specify it manually, or click the Browse button, and select the desired directory in the dialog that opens. Note that the value entered in this field takes precedence over the environment variables.
Gradle JVM	Use this drop-down list to specify JVM under which IntelliJ IDEA will run Gradle when you import the specified Gradle project and when you execute its tasks.

Gradle GroupId Dialog

Use this dialog to specify additional settings for your new Gradle project.

Item	Description
Add as module to	This field appears when you add a new module to the existing project. By default, the field displays the the name of your project. Click to select a different name.
GroupId	Use this field to specify `groupId` of the new project, which will be added to the `build.gradle` file. If a parent Gradle project is specified, this coordinate can be inherited from the parent. To do that, select the Inherit check box.
ArtifactId	Use this field to specify `artifactId` of the new project, which will be added to the `build.gradle` file.
Version	Use this field to specify `version` of the new project, which will be added to the `build.gradle` file. If a parent Gradle project is specified, this coordinate can be inherited from the parent. To do that, select the Inherit check box.

Maven Page

Use this page to define the properties of a new Maven module.

Item	Description

Add as module to	Specify the aggregator Maven project.
	
	If you want the new module to be aggregated in an existing Maven project, click the ellipsis button, and select the desired aggregator. The new Maven Module will be added to the `<modules>` section of the `pom.xml` file of the aggregator, for example:
	
	`<modules>`
	`<module>mod1</module>`
	`<module>mod2</module>`
	`</modules>`
	
	`none` means that the new module will not be aggregated into an existing Maven project.

Parent	Click the ellipsis button and select the desired parent Maven project from the list of existing ones.
	
	If a new module has a parent, the following section is added to its `pom.xml` file, for example:
	
	`<parent>`
	`<groupId>org.something</groupId>`
	`<artifactId>parent1</artifactId>`
	`<version>0.1</version>`
	`</parent>`
	
	`none` means that the new module doesn't have a parent and thus doesn't inherit properties and profiles of any Maven project.

GroupId	Specify `GroupId` of the new module, which will be added to the `pom.xml` file of the module. By default, `GroupId` equals the module name. If a parent Maven project is specified, this coordinate can be inherited from the parent. To do that, select the Inherit check box.
ArtifactId	Specify `ArtifactId` of the new module, which will be added to the `pom.xml` file of the module. By default, `ArtifactId` equals the module name.
Version	Specify `Version` of the new module, which will be added to the `pom.xml` file of the module. If a parent Maven project is specified, this coordinate can be inherited from the parent. To do that, select the Inherit check box.

Add Archetype Dialog

This dialog box is invoked by the Add Archetype button in the Maven page of the New Project wizard. It helps populate the list of archetypes with the ones found by the Maven coordinates and downloaded from the Maven repositories.

Item	Description
GroupId	Specify `GroupId` of the archetype to be sought for in a Maven repository.
ArtifactId	Specify `ArtifactId` of the archetype to be sought for in a Maven repository.
Version	Specify `Version` of the archetype to be sought for in a Maven repository.
Repository (optional)	Optionally, enter the URL of the desired repository.

Maven Settings Page

This page appears when you select Create from archetype option in the project wizard for Maven . Use this page to modify Maven default settings.

Item	Description
Maven home directory	Use this drop-down list to select a bundled Maven version that is available (for Maven2, version 2.2.1 and for Maven3, version 3.0.5) or the result of resolved system variables such as `MAVEN_HOME` or `MAVEN2_HOME` . You can also specify your own Maven version that is installed on your machine. You can click … and select the necessary directory in the dialog that opens .
User settings file	Specify the file that contains user-specific configuration for Maven in the text field. If you need to specify another file, select Override check box, click the ellipsis button and select the desired file in the Select User Settings File dialog.
Local repository	By default, this field shows the path to the local directory under the user home that stores downloads and contains the temporary build artifacts that you have not yet released. If you need to specify another directory, select Override check box, click the ellipsis button and select the desired path in the Select Local Repository dialog.
Properties	By default, the columns in this area display system properties that are passed to Maven for creating a project from the archetype. You can click the following buttons to modify displayed properties: – Add a Maven property. – Remove a Maven property. – Edit a Maven property.

New Project Wizard Android Dialogs

- New Project
- Target Android Devices
- Add an Activity
- Customize the Activity

New Project

Use this page to configure your new project.

Item	Description
Application name	Specify the name of your Android project. By default, IntelliJ IDEA automatically generates the name of the application.
Company Domain	Use this field to specify a qualifier that will be appended to the package name. By default, IntelliJ IDEA automatically generates the name of the qualifier.
Package name	By default, IntelliJ IDEA generate the package name automatically. You can click Edit to modify the specified name of the package.
Project location	Use this field to specify the location of your project. You can click and select the necessary directory in the dialog that opens .

Target Android Devices

IntelliJ IDEA lets select form factors for your Android project.

Item	Description
Phone and Tablet	Select this check box if your target device is either an Android phone or an Android tablet.
Minimum SDK	Use this drop-down list to select the minimum SDK for the Android form factor.
Wear	Select this check box if your target device is the Android Wear.
TV	Select this check box if your target device is the Android TV.
Android Auto	Select this check box if your target device is the Android Auto.
Glass	Select this check box if your target device is the Android Glass.

Add an Activity

IntelliJ IDEA lets you add an activity to your Android project depending on the form factor that you have selected in the Target Android Devices dialog.

Item	Description
Add an activity to Module/Mobile(Auto)	The following Activity templates are available: – Add No Activity – Blank Activity – Empty Activity – Fullscreen Activity – Google AdMob Ads Activity – Google Maps Activity – Login Activity – Master/Detail Flow – Navigation Drawer Activity – Scrolling Activity – Settings Activity – Tabbed Activity
Add an activity to Wear	The following Activity templates are available: – Add No Activity – Always On Wear Activity – Blank Wear Activity – Display Notification – Google Mamps Wear Activity – Watch Face
Add an activity to TV	The following Activity templates are available: – Add No Activity – Android TV Activity
Add an activity to Glass	The following Activity templates are available: – Add No Activity

Customize the Activity

IntelliJ IDEA lets you customize the activity you have selected in the Add an Activity dialog.

Activity Name	**Use this field to specify the name of your activity.**
Layout Name	Use this field to specify the layout name.
Title	Use this field to specify the name of the title.

Note The appearance of other fields, drop-down lists and check boxes depend on the type of form factors and activities you have previously selected.

Create New Project: HTML5 Boilerplate

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose HTML5 Boilerplate from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Version	From this drop-down list, choose the version of the template in accordance to which the stub will be generated. Click to refresh the list of available template versions.

Create New Project: Web Starter Kit

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose Web Starter Kit from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Version	From this drop-down list, choose the version of the template in accordance to which the stub will be generated. Click to refresh the list of available template versions.

Create New Project: React Starter Kit

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose React Starter Kit from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Version	From this drop-down list, choose the version of the template in accordance to which the stub will be generated. Click to refresh the list of available template versions.

Create New Project: Twitter Bootstrap

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose Twitter Bootstrap from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Version	From this drop-down list, choose the version of the template in accordance to which the stub will be generated. Click to refresh the list of available template versions.

Create New Project: Foundation

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose Foundation from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Version	From this drop-down list, choose the version of the template in accordance to which the stub will be generated. Click to refresh the list of available template versions.

Create New Project: Node.js Express App

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose Node.js Express App from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Node interpreter	In this field, specify the location of the Node.js executable file. In most cases, IntelliJ IDEA detects the Node.js executable and fills in the field automatically.
Npm executable	In this field, specify the location of the Node.js package manager file `npm.cmd`. In most cases, IntelliJ IDEA detects the Node.js executable and fills in the field automatically.
Version	From this drop-down list, choose the version of the template in accordance to which the stub will be generated. Click to refresh the list of available template versions.
Template	From this drop-down list, choose the template engine to use. The available options are: – Jade - haml.js successor. – EJS - embedded JavaScript. – Hogan.js. – Handlebars.
CSS	From this drop-down list, choose the CSS to use preprocessor to use. The available options are: – Plain CSS – Stylus – Less – Compass. – Sass.

Create New Project: Meteor Application

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose Meteor App from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Meteor	In this text box, specify the location of the Meteor executable file (see Installing Meteor).
Template	From this drop-down list, choose the sample to generate. To have a basic project structure generated, choose the Default option.
Filename	In this text box, type the name for the mutually related `.js` , `.html` , and `.css` files that will be generated. The text box is available only if the Default sample type is selected from the Template drop-dow list.

Create New Project: PhoneGap/Cordova

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose PhoneGap/Cordova App from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
PhoneGap/Cordova	In this text box, specify the location of the executable file `phonegap.cmd` , or `cordova.cmd` , or `ionic.cmd` (see Installing PhoneGap/Cordova/Ionic).

Create New Project: Yeoman

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose Static Web in the left-hand pane of the Project Category and Options dialog box and then choose Yeoman from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Generators	In this area, create a list of Yeoman generators you need and select the one to use during the current project generation. − To add a generator to the list: 1. Click Install Generator. 2. From the dialog box that opens showing all the available generator packages, select the required package in the left-hand pane and click the Install Generator button that appears in the right-hand pane. You can install several packages one after another without leaving the dialog box. When the installation is over, click Close to return to the list of generators which is already expanded with the newly added package. − To create a project using a generator: 1. In the Location field, specify the folder where the new project will be created. Type the path manually or click the ... and select the folder in the dialog box that opens. 2. If necessary, in the Options text box, specify additional flags to customize the project creation. The set of available flags depends on the selected generator. 3. Select the required generator from the list and click Next. 4. Specify the required settings in the New Project wizard that starts. The number of pages and their contents depend on the chosen generator. 5. On the last page of the wizard, select or clear the Run npm install&bower install check box to specify whether you want to run Node Package Manager and Bower to install the packages that are required for developing the new project. 6. Click Next and choose to open the new project in the current window or in the new one.
Options	In this text box, specify additional flags to customize the project creation. The set of available flags depends on the selected generator.
Configure Node.js and Yeoman	Click this link to open the Yeoman dialog box and change the settings of Node.js and Yeoman.

Create New Project: Composer Project

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose PHP in the left-hand pane of the Project Category and Options dialog box and then choose Composer Project from the list in the right-hand pane.

Use the dialog box to have IntelliJ IDEA create a project with Composer-specific structure. To do this, appoint the Composer instance to use and install the package you need in the project.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
Composer.phar	In this area, appoint the `composer.phar` command definition file to use in project creation. – Use existing composer.phar: choose this option to use commands from a previously downloaded `composer.phar` and specify it location in the text box. – Download composer.phar from getcomposer.org: choose this option to have a new instance of Composer downloaded. The `composer.phar` file will be saved under the project root folder specified in the Location text box.
Package	In this area, specify the package to install during the project creation. – Available packages: from this list box, select the package to install. Use the search field, if necessary: start typing the search string, as you type, the list dynamically reduces to show the packages that match the entered pattern. – Description: this read-only text box briefly explains the functionality of the selected package. – Version to install: from this drop-down list, select the package version. The contents of the list depend on the specific package.
Settings	In this area, specified advanced settings for generating a project stub and installing packages (dependencies): – Path to PHP executable: in this field, specify the location of the PHP engine installation folder. Choose the PHP installation from the list, or type the path manually, or click the Browse button … and choose the desired location in the dialog box that opens. – Command line parameters: in this text box, type the additional command line parameters to adjust the add dependency procedure. For example, to have a package added to the `require-dev` section instead of the default `require` section type `-dev`. For more information about Composer command line options during installation, see https://getcomposer.org/doc/03-cli.md .

Create New Project: Drupal Module

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose PHP in the left-hand pane of the Project Category and Options dialog box and then choose Drupal Module from the list in the right-hand pane.

Use this dialog box to generate and set up a module stub in compliance with the Drupal requirements.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
installation path	In this text box, specify the root folder of the installation. Type the path manually or click the Browse button ... and select the relevant folder in the dialog box that opens.
Set up PHP | Include paths	– Select this check box to have Drupal include paths automatically configured for the project. The following paths will be added to the Include Paths list on the PHP page of the Settings dialog box: 1. `<drupal installation root>/includes` 2. `<drupal installation root>/modules` 3. `<drupal installation root>/sites/all/modules` The include paths are added to the list after you leave the dialog box. – Clear the check box to configure the include paths manually.
Version	From this drop-down list, choose the version of Drupal to use, the supported versions are 6, 7, and 8 .

Create New Project: Google App Engine for PHP

File | new| Project - PHP - App Engine Project

Use this dialog box to configure Google App Engine for PHP project settings.

Item	Description
Application ID	In this text box, type the identifier of your application as you specified it on the Create application page.
SDK directory	In this text box, specify the path to the folder where the Google App Engine SDK for PHP is installed.
Python executable	In this text box, specify the location of the Python executable file. Type the path manually or click the Browse button ... and choose the executable file in the dialog box that opens.

Create New Project: PHP Empty Project

File | New | Project

Welcome Screen | Create New Project

The dialog box opens when you choose PHP in the left-hand pane of the Project Category and Options dialog box and then choose PHP Empty Project from the list in the right-hand pane.

Item	Description
Project Name	In this text box, specify the name of the project, by default it is the name of the folder specified in the location field.
Project Location	In this text box, specify the path to the project folder where the project-related files will be stored.
PHP Language Level	In this drop-down list, specify the PHP functionality scope to get coding assistance for. Each functionality scope is associated with PHP version that supports this functionality. No correlation between the PHP version used in the project and the language level is enforced. Although the language version of each interpreter is detected automatically, you can still tell IntelliJ IDEA to provide you with coding assistance that corresponds to another language level. However, if you attempt to use a code construct that is not supported by the specified language level, IntelliJ IDEA suggests a Switch to PHP <version> quick-fix .
Interpreter	From this drop-down list, choose the PHP interpreter to use in the current project. The list contains all the currently configured local and remote PHP interpreters. See Configuring Local PHP Interpreters and Configuring Remote PHP Interpreters for details.
Include path	This area shows a list of paths to PHP-related items below the PHP home directory. The specified include paths will be used: – By the `require()` , `include()` , `fopen()` , `file()` , `readfile()` , and `file_get_contents()` functions when looking for files to use. – By IntelliJ IDEA when resolving references to included files. Use the Add + and Remove – buttons to manage the contents of the list. Use the Up and Down buttons to change the order of items in the list.

Project Name and Location

Specify the project name, location and, if available, related settings.

Item	Description
Project name	Specify the project name.
Project location	Specify the path to the directory in which you want to create the project. (By default, a directory having the same name as the project is created.) You can click … (Shift+Enter) and select the necessary directory in the dialog that opens . (You can create a new directory in that dialog, e.g. by using .)
More Settings	Click the arrow (or) to show or hide additional settings. Mainly, these are the settings for the module to be created (discussed below). Note that in certain cases those additional settings are unavailable.
Module name	Specify the module name. By default, the project name is used. If you accept this and other default module settings, the project folder will act as the module content root folder . If you change the default module name, IntelliJ IDEA will suggest a new folder (within the project folder) as the module content root folder.
Content root	Specify the path to the module content root folder. (This is where all the files that make up you module will be stored; for more information, see Content Root .) The default path is recommended. To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Module file location	Specify the path to the folder where the `.iml` module file should be created. By default, this file is created in the module content root folder (recommended). To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Project format	Select the project format to be used. (The `.idea` directory-based format is recommended).

New Module Wizard

If a project is open in IntelliJ IDEA, you can access the New Module wizard from:

The main menu: File | New | Module.

The Project tool window : right-click a module folder and select New | Module.

The Project Structure dialog : in the leftmost pane select Modules, above the pane to the right click + and select New Module.

Use the New Module wizard to add a module to your project . (The new module will be created from scratch.)

- Module Category and Options
- J2ME Page
- New Project from Scratch. Android Page
- Maven Page
- Module Name and Location

Module Category and Options

In the left-hand pane, select the module category. This may be the technology that you are going to use, the platform or runtime that your development is going to target, etc.

In the right-hand part of the page, select additional options and specify associated settings.

Note that the set of options you can select from depends on which plugins are currently enabled in IntelliJ IDEA.

- Java
- Java Enterprise
- J2ME
- Clouds
- Spring
- IntelliJ Platform Plugin
- Android
- Maven
- Gradle
- Groovy
- Grails
- Griffon
- Static Web
- Flash

Java

Select this option if you are going to use your new module to develop a Java application.

Item	Description
Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)

If necessary, select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

Java Enterprise

Select this option if you are going to use your new module to develop a Java EE application.

Item	Description
Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Java EE version	Select the Java EE version to be supported. (Affects the corresponding version setting for the Web Application, EJB and JavaEE Application options.)
Application Server	Specify the application server that you are going to use to deploy and run your application. As a result, IntelliJ IDEA will create a run/debug configuration for the specified server. (You can specify the server later.) You can select a server which IntelliJ IDEA is already aware of, or specify another "new" server. To specify a new server, click New and select the server of interest. Then, specify the server settings: – For a server installed locally, specify the path to the server installation directory. (Click … to select the directory in the corresponding dialog .) – For a hosted server (Cloud Foundry or CloudBees), specify your user account details.

Select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

J2ME

Select this option if you are going to use your new module to develop for Java ME .

Item	Description
Module SDK	Specify an SDK for your module. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and, in the dialog that opens , select the installation folder of the desired Java ME SDK. (By this time, the corresponding SDK must already be installed on your computer.)
SQL Support	Select the check box to enable SQL support. Select the SQL dialect to be used by default from the list.

Clouds

Select this option if you are going to deploy your application to a cloud platform such as CloudBees , Cloud Foundry , Heroku or OpenShift . See also, Working with Cloud Platforms .

Item	Description
Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list.

	Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Account	Specify your cloud user account. If the corresponding user account is already registerd in IntelliJ IDEA, select it from the list. Otherwise, click New, select the cloud platfrom and specify your user account settings in the dialog that opens.
Application	Cloud platform-specific application settings. CloudBees and Cloud Foundry. IntelliJ IDEA will create a sample Java web application which you'll be able to deploy to the cloud and run straight away. – Version. The version of the Servlet specification to be supported. – Create web.xml. For version 3.0 or later: select this check box to create the deployment descriptor file `web.xml` . (For earlier versions, this file is always created.) Heroku. You can select to create a new application or to git-clone the source code for one of your applications already deployed on Heroku. – Template. A new sample application will be created. You'll be able to deploy this application to Heroku straight away. – Existing. Select the application whose source code you want to clone. OpenShift. You can select to git-clone the source code for one of your applications already deployed OpenShift or to create a new application. – Existing. Select the application whose source code you want to clone. – New. Select this option to create a new application. Specify the settings for your new application (for OpenShift terminology, see OpenShift documentation): – Standalone Cartridge. Select the primary cartridge. – Gear size. Select the gear size. – Scaling. Select the check box if the application should be scalable. – Embeddable Cartridges. Select (additional) embedded cartridges.

Spring

Select this option if you are going to use your new module to develop a Spring application.

Item	Description
Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)

For information on other options and settings, see:

– Spring

– Spring MVC, Spring Batch, or other Spring framework

– Additional Libraries and Frameworks

See also, Enabling Spring Support .

IntelliJ Platform Plugin

Select this option if you are going to use your new module to develop a plugin for IntelliJ IDEA or other IntelliJ Platform-based IDE.

Item	Description
Module SDK	Specify an SDK for your module. If the necessary SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and, in the dialog that opens , select the installation folder of the desired IntelliJ IDEA version. (An IntelliJ IDEA installation acts as an IntelliJ Platform Plugin SDK.) (By this time, the corresponding IntelliJ IDEA version must already be installed on your computer.)
Groovy	Select the check box to be able to use Groovy . Specify the Groovy installation to be used. Use library. If the desired version of Groovy is already defined in IntelliJ IDEA, select it from the list. (Groovy in IntelliJ IDEA is represented by a library .) Create. Click this button to create a library for Groovy. In the dialog that opens , select the Groovy installation directory.
SQL Support	Select the check box to enable SQL support. Select the SQL dialect to be used by default from the list.

Android

Select this option if you are going to use your new module to develop for the Android OS. Select:

– Phone & Tablet Module to develop an Android application for an Android phone or an Android tablet.

– Android Wear Module to develop an Android application for the Android Wear.

– Android Library Module to develop a shared Android library.

– Android TV Module to develop an Android application for the Android TV.

– Glass Module to develop an Android application for the Android Glass.

– Import Gradle Project to import an existing Gradle project as a module.

– Import Eclipse ADT Project to import an existing Eclipse ADT project as a module .

– Import .JAR/.AAR Package to import an existing JAR or AAR package as a new module.

– Java Library to create a new Java library.

Maven

Select this option if you are going to use your new module to develop a Java application with dependencies managed by Maven .

Item	Description

Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Create from archetype	If this check box is not selected, the new `pom.xml` file will contain the basic information. If this check box is selected, the new module will be created on the base of a Maven archetype chosen from the list that includes both the standard archetypes, and the ones found in Maven indices. You can modify Maven properties on Maven Settings Page . If you want to populate the list with some archetype from a remote Maven repository, click the Add Archetype button, and find the desired archetype by Maven coordinates specified in Add Archetype Dialog .

Gradle

Select this option if you want a module with a Gradle build script (`build.gradle`) to be created.

Item	Description
Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)

Groovy

Select this option if you are going to develop a Groovy application.

Item	Description
Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Groovy library	If the desired version of Groovy is already defined in IntelliJ IDEA, select it from the list. (Groovy in IntelliJ IDEA is represented by a library .) Create. Click this button to create a library for Groovy. In the dialog that opens , select the Groovy installation directory.

If necessary, select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

Grails

Select this option if you are going to develop a Grails application.

See also Creating a Grails Application Module .

Item	Description
Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Grails library	If the desired version of Grails is already defined in IntelliJ IDEA, select it from the list. (Grails in IntelliJ IDEA is represented by a library .) Create. Click this button to create a library for Grails. In the dialog that opens , select the Grails installation directory.

If necessary, select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

Griffon

Select this option if you are going to develop a Griffon application.

See also Creating a Griffon Application Module .

Item	Description
Module SDK	Specify an SDK (JDK) for your module. If the necessary JDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select JDK. Then, in the dialog that opens , select the installation folder of the desired JDK. (By this time, the corresponding JDK must already be installed on your computer.)
Griffon library	If the desired version of Griffon is already defined in IntelliJ IDEA, select it from the list. (Griffon in IntelliJ IDEA is represented by a library .) Create. Click this button to create a library for Griffon. In the dialog that opens , select the Griffon installation directory.

If necessary, select additional options and specify associated settings. For more information, see Additional Libraries and Frameworks .

Static Web

Select Static Web if you are going to develop a Web application using HTML / CSS , JavaScript , PHP and related frameworks.

The other options are for developing a Web site or a Web front end using an HTML5 Boilerplate template, or the Foundation or the Bootstrap framework.

Flash

Select this option if you are going to use your new module to develop for the Adobe Flash runtimes using Flex or ActionScript . Specify the associated settings.

Item	Description
Target platform	Select the target environment for the content that you are going to develop: – Web for Flash player / Web browser-targeted content. – Desktop for Adobe AIR-targeted content. – Mobile for the content intended for mobile devices (Android, iOS, etc.).
Pure ActionScript	Select this check box if you are not going to use MXML (i.e. all your source code will be written in ActionScript).
Output type	Select the intended output type, that is, what your resulting content is going to be: – Application. A runnable application, an SWF file. – Runtime-loaded module. A dynamically-loadable module , an SWF file. – Library. An SWC file.
Target devices	For a Mobile Application: use the Android and iOS check boxes to specify the intended target devices for your application. As a result, IntelliJ IDEA enables or disables creating an application descriptor and packaging your application for the corresponding devices. (The Android and iOS check boxes on this page correspond to the Enabled check boxes on the Android and iOS tabs in the build configuration that will be created.)
Flex/AIR SDK	Select the Flex or AIR SDK to be used. If the list is empty or does not contain the required SDK, click … (Shift+Enter) and add the required SDK in the Configure SDK dialog.
Target player	For the Web target platform: the target Flash Player version (readonly). (This setting is defined by the selected Flex SDK version.)
Create sample app	For the Application output type: select this check box if you want a sample application to be created. You can use this sample application for learning and also as a basis for your own application development. If necessary, change the source file name suggested by IntelliJ IDEA.
Create HTML wrapper template	For a Web Application: select this check box if you want an HTML wrapper template for your application to be created. Select or deselect the associated options as needed: – Enable integration with browser navigation . Select this option to enable deep linking. Deep linking lets users navigate their interactions with the application by using the Back and Forward buttons in their browser. – Check Flash player version . If you select this option, the compiled application will check for the correct version of Flash Player. – Express install . If you select this option, the application will run an SWF file in the existing Flash Player to upgrade users to the latest version of the player.

Module Name and Location

Specify the module name and location.

Item	Description
Module name	Specify the module name.
Content root	Specify the path to the module content root folder. (This is where all the files that make up you module will be stored; for more information, see Content Root .) By default, a folder having the same name as the module is created in the project directory (recommended). To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Module file location	Specify the path to the folder where the `.iml` module file should be created. By default, this file is created in the module content root folder (recommended). To use a different folder, click … (Shift+Enter) and select the necessary folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)

Tool Windows Reference

View | Tool Windows

The Tool Windows Reference contains detailed information about the functionality, controls and menus of the IntelliJ IDEA tool windows .

In this section:

- Messages (Alt+0)
- Project (Alt+1)
- Favorites (Alt+2)
- Find (Alt+3)
- Run (Alt+4)
- Debug (Alt+5)
- TODO (Alt+6)
- Structure (Alt+7)
- Hierarchy (Alt+8)
- Changes (Alt+9)
- Dependency Viewer
- Inspection
- Version Control
- Ant Build
- Database
- Phing Build
- Duplicates
- Maven Projects
- Module Dependencies
- GUI Designer Palette
- DSM
- REST Client

Ant Build Tool Window

View | Tool Windows | Ant Build

Use this tool window to add Ant build scripts to your project, control behavior of the build, and run build targets.

– Prerequisites

– Ant Build

– Ant build results

Prerequisites

Before you start working with Ant Build tool window, make sure that the Ant support plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Ant Build

Item	Description
	Click this button to add an Ant build file to the current project.
	Click this button to remove the reference to the selected build file from the current project.
	Click this button to run the selected build target.
	When this button is pressed, only the primary targets are visible.
	Use these buttons to expand or collapse all the nodes.
	Click this button to show the properties of the selected Ant build file .
	Click this button to show the corresponding reference topic.

Ant build results

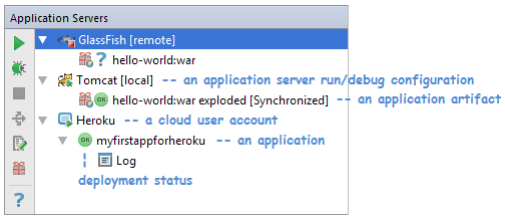
Results of running Ant targets or the entire builds are shown in the Messages tool window .

Application Servers tool window

View | Tool Windows | Application Servers

For this tool window to be available, there must be a server run/debug configuration in your project , or a cloud user account must be registered in IntelliJ IDEA.

The Application Servers tool window lets you manage your applications on application servers and cloud platforms. You can start and stop server run/debug configurations and connect to cloud platforms, deploy and undeploy your application artifacts as well as perform other, associated tasks.



All the available functions are accessed by means of the toolbar icons and context menu commands.

- Icons and commands for server run configurations
- Icons and commands for server artifacts
- Icons and commands for cloud user accounts
- Icons and commands for cloud apps
- Deployment status icons

See also, Working with Server Run/Debug Configurations and Working with Cloud Platforms .

Icons and commands for server run configurations

Icon	Command	Description
	Run/Connect	Start the selected run/debug configuration in the run mode. For a local configuration, normally, the corresponding server will be started. For a remote configuration, IntelliJ IDEA will connect to the server.
	Debug	Start the selected run/debug configuration in the debug mode.
	Stop/Disconnect	Stop the selected run/debug configuration. For a local configuration, normally, the corresponding server will be stopped. For a remote configuration, IntelliJ IDEA will disconnect from the server.
	Deploy All	Deploy all the artifacts associated with the selected run/debug configuration.
	Edit Configuration	Edit the settings for the selected run/debug configuration.
	Artifacts	Edit the deployment list for the selected run/debug configuration. (The Artifacts to Deploy Dialog will open.)

Icons and commands for server artifacts

Icon	Command	Description
	(Re)deploy	Deploy or redeploy the selected artifact.
	Undeploy	Undeploy the selected artifact.
	Remove	Remove the selected artifact from the corresponding deployment list and undeploy the artifact from the server.

Icons and commands for cloud user accounts

Icon	Command	Description
	Connect	Connect (log on) to the corresponding cloud platform.
	Disconnect	Disconnect (log off) from the corresponding cloud platform.
	Edit Configuration	Edit your cloud user account settings.
	Deploy	Deploy your app by means of a cloud deployment run/debug configuration.
	Debug	Deploy your app and start debugging it by means of a cloud deployment run/debug configuration.

Icons and commands for cloud apps

Icon	Command	Description
	(Re)deploy	Deploy or redeploy the selected app.
	Undeploy	Undeploy the selected app.
	Debug	Start debugging the selected application.
	Edit Configuration	Edit the settings for an associated cloud deployment run/debug configuration.

Deployment status icons

Icon	Status
	Unknown
	Deployed
	Undeployed

Bean Validation Tool Window

To open this tool window:

View | Tool Windows | Bean Validation

See also, Showing a tool window.

Note that this tool window is available only if there is a library that implements Bean Validation in the dependencies of one or more of your modules .

The Bean Validation tool window provides a categorized hierarchical view of your bean validation resources. These include the appropriately annotated Java sources as well as xml validation descriptors and constraint mappings.

At the top of the hierarchy are your modules. One level below are categories.

The main categories are constraints, constraint validators and constraints mappings.

You can show or hide the categories. You can also open the elements shown in the tool window in the editor.

- Showing and hiding categories
- Opening elements in the editor

Showing and hiding categories

Use the toolbar buttons to show or hide the corresponding category.

ItemTooltip	Description	
	Constraints	Click this button to show or hide the elements related to bean validation constraints. Shown in this category are the constraints (constraint annotations) defined in your source code ( ) and also the ones available in the bean validation library that you are using ( ). For each of the constraints, the related elements are shown (if any). These may be the corresponding constraint validators or other entities that reference the constraint.
	Validators	Click this button to show or hide bean validators. Shown in this category are the validators for which you developed the source code yourself ( ) and also the ones available in the bean validation library that you are using ( ).
	Constraint Mappings	Click this button to show or hide xml constraint mappings. Shown in this category are the xml files that contain the corresponding mappings. For each of the files, a structured view of mapping definition elements is provided.

Opening elements in the editor

You can open the elements shown in the Bean Validation tool window in the editor. To do that, select the element of interest and press F4 .

To open the elements which are the "leaves" of the tree (i. e. the ones at the bottom of the hierarchy), you can also use a double click.

CDI Tool Window

To open this tool window:

View | Tool Windows | CDI

See also, Showing a tool window.

Note that this tool window is available only if there is a library that implements CDI in the dependencies of one or more of your modules .

The CDI tool window provides a categorized hierarchical view of your CDI resources (beans).

At the top of the hierarchy are your modules. One level below are categories.

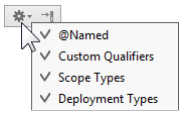
The categories correspond to bean types and qualifiers.

You can show or hide the categories. You can also open the elements shown in the tool window in the editor.

- Showing and hiding categories
- Opening elements in the editor

Showing and hiding categories

To show or hide a category, click on the title bar, and then click the necessary option.



The following options are available:

- @Named. Show or hide @Named beans (the elements annotated with `@Named`).
- Custom Qualifiers. Show or hide the elements annotated with custom annotations.
- Scope Types. Show or hide scope types (the elements annotated with `@ApplicationScoped`, `@SessionScoped`, `@RequestScoped`, etc.).
- Deployment Types. Show or hide deployment types.

Opening elements in the editor

You can open the elements shown in the CDI tool window in the editor. To do that, select the element of interest and press `F4`.

To open the elements which are the "leaves" of the tree (i. e. the ones at the bottom of the hierarchy), you can also use a double click.

Command Line Tools Console Tool Window

Tools | Run Command

Ctrl+Shift+X

The tool window is available only when the Command Line Tool Support plugin is downloaded, installed, and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Use the tabs of this tool window to type and run commands in the command line and to validate tool definition files for structure consistence.

The character set to be used in the tool window is chosen from the Console encoding drop-down list on the Command Line Tool Support page of the Settings dialog box.

The tool window consists of the following tabs and areas:

- The Input text box is displayed in the bottom of the tool window if you have chosen Tool window in the Show console in area of the Command Line Tool Support page. Otherwise, the Command Line Tools Input pane opens in a separate pop-up window.
  In this text box, type the desired command in the format `<tool alias> <command>`
- The output tab shows the results of executing commands. The tab is named after the last invoked command.
- The Tool definition file errors tab is not shown by default. The tab is accessible only if any structure discrepancies are detected in the tool definition file.
  Every time you invoke a command, IntelliJ IDEA performs full validation of the tool definition file. If the validation fails, IntelliJ IDEA displays a Command Line Tool pop-up window with a notification on validation failure. Upon clicking the More link, the Tool definition file errors tab opens showing messages on detected inconsistencies. Each message contains information on the file and the line number where the problem was found, as well as a brief description of the error.

  You can close the tab by clicking the cross on its header. To re-open the tab, again click More in the Command Line Tool notification pop-up window, which remains on the screen until you close it manually.

  Upon validation failure, the tool is marked with the Invalid description icon in the Command Line Tool Support page.

Toolbar Options

Item	Tooltip and Shortcut	Description
	Stop Ctrl+F2	Click this button to cancel execution of a command without closing the tool window.
	To previous command Ctrl+Alt+Up	Click this button to navigate to the previous command.
	To next command Ctrl+Alt+Down	Click this button to navigate to the next command.
	Use Soft Wraps	Click this toggle button to have the soft wrap mode applied to the output.
	Scroll to the end	Click this button to navigate to the bottom of the output tab named after the last invoked command.
	Print	Click this button to have the contents of the console printed out. Upon clicking the button, IntelliJ IDEA opens the Print dialog box, where you can configure the printing procedure and output.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.
	Export to Text Alt+O	Click this button to have the results of executing commands saved in a text file. In the Export Preview dialog box, that opens, specify the target file, and click Save.
	Ctrl+Shift+F4	Click this button to close the tool window. If one or more commands are still running, the Command Line Tool dialog box opens. Specify whether you want to stop them or leave running in the background by clicking one of the following buttons: – Terminate and close – Close without terminating
	Help F1	Use this icon or shortcut to open the corresponding help page.

Command Line Tools Input Pane

Tools | Run Command

`Ctrl+Shift+X`

In this input pane, type the desired command in the format `<tool alias> <command>`

The input pane is available in two modes:

- As a separate pop-up window, if you have chosen Tool window in the Show console in area of the Command Line Tool Support page.
- As the Input text box at the bottom of the Command Line Tools tool window.

View the results of command execution in the output tab of the dedicated Command Line Tools Console tool window. The tab is named after the last invoked command.

Coverage Tool Window

View | Tool Windows | Coverage

This tool window appears on running with coverage , and displays coverage measurement results.

In this section:

– Toolbar

– Context menu

Toolbar

Item	Description
	Click this button to go up one level.
	When this button is pressed, all the packages are displayed as a single-level view.
	When this button is pressed, source code of the class selected in the tool window, automatically opens in a separate editor tab, and gains the focus.
	When this button is pressed, when source code of certain class gets the focus in the editor, the corresponding node is automatically highlighted in the tool window.

	Click this button to generate a code coverage report and save it to the specified directory. See Generating Code Coverage Report for details.
	The button is not available when the tests are executed on Karma because a coverage report is actually generated on the disk every time Karma tests are run. The format of a coverage report can be configured in the configuration file, for example:
	`// karma.conf.js`
	`module.exports = function(config) {`
	`config.set({ ...`
	`// optionally, configure the reporter`
	`coverageReporter: { type : 'html', dir : 'coverage/' }`
	`...`
	`});};`
	The following `type` values are acceptable:
	– `html` produces a bunch of HTML files with annotated source code
	– `lcovonly` produces an `lcov.info` file
	– `lcov` produces HTML + `.lcov` files. This format is applied by default.
	– `cobertura` produces a `cobertura-coverage.xml` file for easy Hudson integration
	– `text-summary` produces a compact text summary of coverage, typically to the console
	– `text` produces a detailed text table with coverage for all files
	This toolbar button is duplicated by the main menu command Analyze | Generate Coverage Report.

	Click this button to close the tool window. This toolbar button is duplicated by the main menu command Analyze | Hide Coverage Data.
	Click this button to show reference.

Context menu

Item	Shortcut	Description
Jump to Source	F4	Choose this command to open the selected file in the editor.

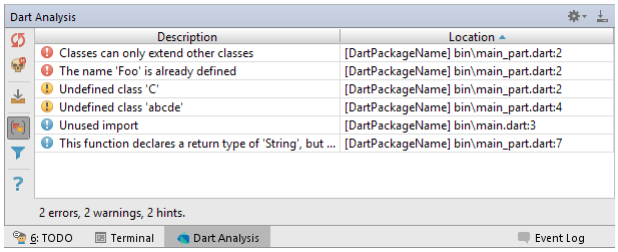
Dart Analysis Tool Window

View | Tool Windows | Dart Analysis

`Alt+0`

The tool window is available only when the Dart plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

IntelliJ IDEA provides interface for using the Dart Analyzer tool . The tool is intended for static analysis of your Dart source code. All inconsistencies and potential problems are reported in the dedicated Dart Analysis tool window with the possibility to navigate to the fragment of the source code where the problem was detected.



On this page:

- Messages List
- Grouping and Sorting
- Toolbar Buttons
- Context Menu

Messages List

All the problem reports are displayed in a table which consists of two columns, Description and Location, and a summary below.

- The Description column shows the message itself, which can be an error message, a warning, or a hint.
- The Location column shows the following data:
  - The name of the Dart package in which the problem arose. The Dart package name corresponds to the `project name` from `pubspec.yaml` .
  - The relative from the the Dart package root to the file where the problem arose.
  - The line number where the problem arose.
- The Summary area below the table shows the number of detected errors, warnings, or hints and the filter status.

Grouping and Sorting

The messages in the Dart Analysis tool window can be grouped and sorted. By default, the messages are primarily sorted by their severity, that is, the error messages are shown at the top of the list, then come warnings, and finally hints are displayed. In each severity group, messages are grouped by the Dart package name, which is super handy for projects with multiple `pubspec.yaml` files. In each package group, the messages are grouped by their file paths. Finally, in each file group, the problems are sorted by the line number where they occurred.

You can re-configure the secondary sorting:

- Click the Location column header to switch between the ascending and descending order of the Dart package name + file path sorting. Note that this does not affect the order of severity groups, that is, errors are always shown first. Line numbers are not taken into consideration either, for each particular file problems are always shown in ascending order of line numbers.
- Click the Description column header to perform the secondary sorting alphabetically by the problem description. Note that this does no affect the primary grouping by severity.

To disable the primary grouping by severity, release the Group by Severity toggle button on the toolbar. After that the severity of a problem is not taken into consideration at all and problem reports are sorted only by their Description or Location (Dart package name + file path). This sorting method guarantees that all the problems for each particular file are grouped together in the table, regardless of their severity.

Toolbar Buttons

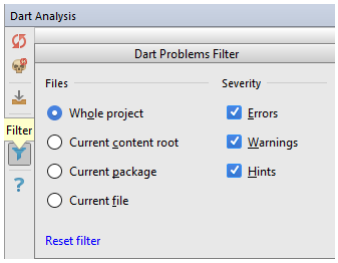
Item	Tooltip and shortcut	Description
	Reanalyze Dart Sources	Click this button to run the analysis of the Dart source code of the project wothout stopping the Dart Analysis server.
	Restart Dart Analysis Server	Click this button to kill the Dart Analysis server and then start it.
	Autoscroll to Source	If this button is pressed, the file that contains the selected error automatically opens in the editor, with the caret at the appropriate line.
	Group by Severity	– When this toggle button is pressed, the error messages are shown at the top of the list, then come warnings, and finally hints are displayed. In each severity group, messages are grouped by the Dart package name, which is super handy for projects with multiple `pubspec.yaml` files. In each package group, the messages are grouped by their file paths. Finally, in each file group, the problems are sorted by the line number where they occurred. You can re-configure the secondary sorting: – Click the Location column header to switch between the ascending and descending order of the Dart package name + file path sorting. Note that this does not affect the order of severity groups, that is, errors are always shown first. Line numbers are not taken into consideration either, for each particular file problems are always

shown in ascending order of line numbers.
- Click the Description column header to perform the secondary sorting alphabetically by the problem description. Note that this does no affect the primary grouping by severity.

- Release this toggle button to disable the primary grouping by severity. After that the severity of a problem is not taken into consideration at all and problem reports are sorted only by their Description or Location (Dart package name + file path). This sorting method guarantees that all the problems for each particular file are grouped together in the table, regardless of their severity.

Filter

Click this button to open the Dart Problems Filter pop-up window where you can configure the criteria according to which a problem report is displayed in the tool window or not.



- In the Severity area, specify the types of messages to be shown, the available types are Errors, Warnings, and Hints. To have the problems of a severity level displayed, select the check box next to this severity level.
- In the Files area, choose the scope for which you want to see problem reports. The available options are:
  - Whole project
  - Current content root
  - Current package: when this option is selected but no `pubspec.yaml` file is detected up the folder hierarchy starting from the current file, then filtering is performed according to the content root where current file is located.
  - Current file
- Click the Reset all filters link to restore the default filter settings.

Filtering is applied immediately as soon as you change the current settings.

Help

Use this button to navigate to the help topic for the tool window.

Context Menu

Item	Shortcut	Description
Jump to source	F4	Click this button to navigate to the fragment of code that caused the selected problem.
Copy	Ctrl+C	Take the line at caret to the clipboard.

Database Console

From the Database tool window (for any node within a DB data source):

– on the title bar if the toolbar is hidden

– on the toolbar if the toolbar is shown

– Open Console or Open New Console from the context menu

– `Ctrl+Shift+F10`

From the Scratches view of the Project tool window :

– View | Jump to Source

– Jump to Source from the context menu

– `F4`

On this page:

– Overview

– Input pane

– Toolbar of the Database Console tool window

– Output pane

– Result pane

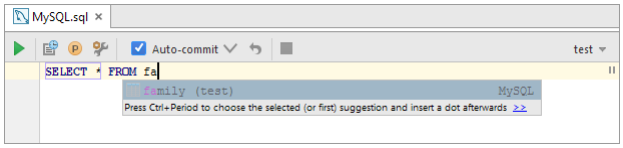
– Parameters pane

See also, Working with Database Consoles .

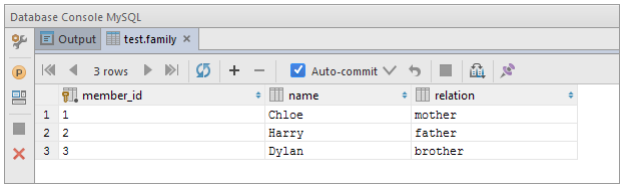
Overview

Database consoles let you compose and execute SQL statements for databases defined in IntelliJ IDEA as data sources. They also let you analyze and modify the retrieved data.

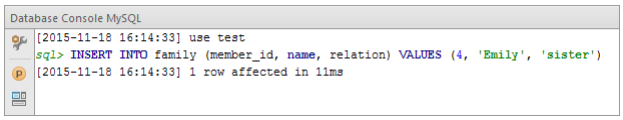
The input pane of a database console opens as a separate editor tab. This is where you compose your SQL statements.



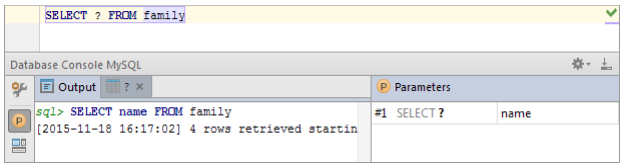
When you execute your first statement ( ), the Database Console tool window opens. If the executed statement retrieves data (e.g. `SELECT`), there are two panes in the tool window shown on the Output and the Result tabs. (The tab showing retrieved data may be labeled Result # or, if appropriate, the table name may be shown.)



Otherwise, only the output pane is shown.



Additionally, you can open the Parameters pane ( ) to manage parameters in SQL statements.



Input pane

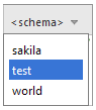
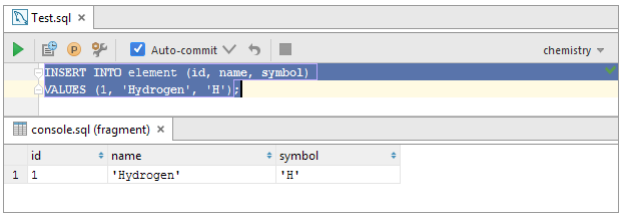
Use the input pane to compose and execute your SQL statements as well as to perform other, associated tasks.

The available functions are accessed by means of the toolbar icons, keyboard shortcuts and context menu commands.

– Toolbar icons and shortcuts

– Most useful context menu commands

Item	Shortcut	Description
Toolbar icons and shortcuts		
Execute	`Ctrl+Enter`	Use this icon or shortcut to execute the selected (highlighted) SQL statement or statements. If nothing is selected, the current statement is executed. See also, Run 'console.sql', Execute in Console and Executing an SQL statement.

Browse Console History	Ctrl+Alt+E	Use this icon or shortcut to open a dialog that shows all the statements that you have run for the corresponding data source. See also, Executing auto-memorized statements.
View Parameters		Use this icon to open or close the Parameters pane .
Settings		Use this icon to open the Database page of the Settings dialog to view or edit the settings for the database, Hibernate and JPA consoles, the Table Editor and the Database tool window.
Auto-commit		Use this check box to turn the autocommit mode for the database connection on or off. In the autocommit mode, each SQL statement is executed in its own transaction that is implicitly committed. Consequently, the SQL statements executed in this mode cannot be rolled back. If the autocommit mode is off, transactions are committed or rolled back explicitly by means of the commit or rollback command. Each commit or rollback starts a new transaction which provides grouping for a series of subsequent SQL statements. In this case, the data manipulations in the transaction scope are committed or rolled back all at once when the transaction is committed or rolled back.
Commit		If the autocommit mode is off, use this icon to commit the current transaction.
Rollback		If the autocommit mode is off, use this icon to roll back the current transaction.
Cancel Running Statements	Ctrl+F2	Use this icon or shortcut to terminate execution of the current statement or statements.
<schema>		A list for selecting the default schema or database (may be unavailable). 
Most useful context menu commands		
Open DML Table Editor		If an `INSERT` statement is currently selected: open the editor for working with the data in table format.  See also, Editing data for INSERT statements in table format.
Change Dialect (<CurrentDialect>)		Use this command to change the SQL dialect being used. Select the necessary dialect from the list. In addition to particular dialects, also the following option is available: – <Generic>. Basic SQL92-based support is provided including completion and highlighting for SQL keywords, and table and column names. Syntax error highlighting is not available. So all the statements in the input pane are always shown as syntactically correct. See also, Changing the SQL dialect.
Explain Plan		Show an execution plan (a.k.a. explain plan) for the current statement. The result is shown in a mixed tree/table format on a dedicated Plan tab.
Explain Plan (Raw)		Show an execution plan (a.k.a. explain plan) for the current statement. The result is shown in table format. (Technically, `EXPLAIN <CURRENT_STATEMENT>` or similar statement is executed.)
Execute	Ctrl+Enter	Execute the current statement or the sequence of selected statements.
Execute to File		Execute the current statement (e.g. `SELECT`) and save the result in a text file. Select the output format, and specify the file location and name.
Run 'console.sql'	Ctrl+Shift+F10	Use this command or shortcut to execute all the statements contained in the console.
Diagrams	Ctrl+Shift+Alt+U Ctrl+Alt+U	If the cursor is within the name of a schema or table: open a UML class diagram for the schema or table. – Show Visualisation (Ctrl+Shift+Alt+U). The diagram opens on a separate editor tab. – Show Visualisation Popup (Ctrl+Alt+U). The diagram opens in a pop-up window.

Toolbar of the Database Console tool window

To hide or show the toolbar, click on the title bar and select Show Toolbar.

Item	Shortcut	Description
Settings		Use this icon to open the Database page of the Settings dialog to view or edit the settings for the database, Hibernate and JPA consoles, the Table Editor and the Database tool window.
Enable SYS.DBMS_OUTPUT	Ctrl+F8	For Oracle: use this icon or shortcut to enable or disable showing the contents of the DBMS_OUTPUT buffer in the output pane.
View Parameters		Use this icon to open or close the Parameters pane .
Restore Layout		Use this icon to restore the original tool window layout (after the rearrangements that you have made).
Cancel Running Statements	Ctrl+F2	Use this icon or shortcut to terminate execution of the current statement or statements.
Close	Ctrl+Shift+F4	Use this icon or shortcut to close the tool window.

Output pane

This pane shows the SQL statements that you have run as well as information about other operations performed in the console. These include turning the autocommit mode on or off, committing or rolling back a transaction, etc.

The information about the errors that occur is also shown in this pane.

For most of the events the following information is provided:

- Timestamp, that is, when the event took place.
- For data definition and data manipulation operations - how many rows were affected (e.g. added, changed or deleted). For data retrieval operations - how many rows were retrieved.
- Duration in milliseconds.

Use the following context menu commands:

- Copy (`Ctrl+C`) to copy the text selected in the output pane to the clipboard.
- Compare with Clipboard to compare the text selected in the output pane with the contents of the clipboard.
- Clear All to clear all the contents of the output pane.

Result pane

This pane shows the data retrieved from the database in table format. You can sort, add, edit and remove the data as well as perform other, associated tasks.

- Main functions
- Using the header row

Main functions. Most of the functions in the Result pane are accessed by means of controls on the toolbar, context menu commands for the data cells, and associated keyboard shortcuts.

Item	Shortcut	Description
and		These icons and corresponding commands are for switching between the result set pages, i.e. the pages that show the retrieved data. A fixed number of rows shown simultaneously is referred to as a result set page. If this number is less than the number of rows that satisfy the query, only a subset of all the rows is shown at a time. In such cases, you can use and to switch between the subsets. (If all the rows are currently shown, these icons and the corresponding commands are inactive.) The result set page size is set on the Database page of the Settings dialog.
First Page		Use this icon or command to switch to the first of the result set pages to see the first series of rows.
Previous Page	`Ctrl+Alt+Up`	Use this icon, command or shortcut to switch to the previous result set page to see the previous series of rows.
Next Page	`Ctrl+Alt+Down`	Use this icon, command or shortcut to switch to the next result set page to see the next series of rows.
Last Page		Use this icon or command to switch to the last of the result set pages to see the last series of rows.
Reload Page	`Ctrl+F5`	Use this icon, command or shortcut to refresh the current table view. Use this function to: - Synchronize the data shown with the actual contents of the database. - Apply the Result set page size setting after its change.
Add New Row	`Alt+Insert`	Use this icon, command or shortcut to add a new row to the table. Complete entering a value into a cell by pressing `Enter` . To save the new row, select Submit New Row from the context menu or press `Ctrl+Enter` . If inappropriate in the current context (i.e. for the table currently shown), this function is not available. See also, Adding a row.
Delete Selected Rows	`Ctrl+Y`	Use this icon, command or shortcut to delete the selected row or rows. Rows are selected by clicking the cells in the column where the row numbers are shown. To select more than one row, use mouse clicks in combination with the `Ctrl` key. If inappropriate in the current context (i.e. for the table currently shown), this function is not available.
Auto-commit		Use this check box to turn the autocommit mode for the database connection on or off. In the autocommit mode, each SQL statement is executed in its own transaction that is implicitly committed. Consequently, the SQL statements executed in this mode cannot be rolled back. (Each change of a value in the table, or adding or deleting a row results in executing an SQL statement.) If the autocommit mode is off, transactions are committed or rolled back explicitly by means of the commit or rollback command. Each commit or rollback starts a new transaction which provides grouping for a series of subsequent SQL statements. In this case, the data manipulations in the transaction scope are committed or rolled back all at once when the transaction is committed or rolled back.
Commit		If the autocommit mode is off, use this icon or command to commit the current transaction.
Rollback		If the autocommit mode is off, use this icon or command to roll back the current transaction.
Cancel Query	`Ctrl+F2`	Use this icon or shortcut to terminate execution of the current query.
Pin Tab		Use this icon or command to pin the tab to the tool window to keep the query result. See also, Pinning the Result tab.

Tab S...(TSV) Data Extractor		Use this button or command to open a menu in which you can select an output format for your data. The Configure CSV Formats command in that menu opens the CSV Formats Dialog that lets you manage the delimiter-separated values formats (e.g. CSV, TSV).
View Query		Use this button to view the query which was used to generate the table. To close the pane where the query is shown, press Escape.
		This icon provides access to the following commands: – Transpose. Use this command to turn the transposed table view on or off. (In the transposed view, the rows and columns are interchanged. So, the rows are shown as columns and vice versa.) – Reset View. Use this command to restore the initial table view after reordering or hiding the columns, or sorting the data. – Settings. Use this command to open the Database page of the Settings dialog to view or edit the settings for the Database, Hibernate and JPA consoles, the Table Editor and the Database tool window.
Clone Row	Ctrl+D	Use this command or shortcut to create a copy of the selected row.
Edit Value	F2	Use this command or shortcut to start editing a value in the selected table cell. (Alternatively, you can double-click the cell or simply start typing.) To save the modified value, press Enter. To cancel editing, press Escape. See also, Modifying cell contents.
Quick Documentation	Ctrl+Q	Use this command or shortcut to open the quick documentation view. To close the view, press Escape. For more information, see Using the quick documentation view.
Go To | Row	Ctrl+G	Use this command or shortcut to switch to a specified row. In the dialog that opens, specify the row number to go to.
Go To | Related Data	F4	Use this command or shortcut to switch to a related record. The command options are a combination of those for Go To | Referenced Data and Go To | Referencing Data . The command is not available if there are no related records.
Go To | Referenced Data	Ctrl+B	Use this command or shortcut to switch to a record that the current record references. If more than one record is referenced, select the target record in the pop-up that appears. The command is not available if there are no referenced records.
Go To | Referencing Data	Alt+F7	Use this command or shortcut to see the records that reference the current record. In the pop-up that appears there are two categories for the target records: – First Referencing Row. All the rows in the corresponding table will be shown and the first of the rows that references the current row will be selected. – All Referencing Rows. Only the rows that reference the current row will be shown. The command is not available if there are no records that reference the current one.
Dump Data | To Clipboard		Use this command to copy the table data onto the clipboard.
Dump Data | To File		Use this command to save the table data in a file. In the dialog that opens, specify the location and name of the file.
Save LOB		Use this command to save the large object (LOB) currently selected in the table in a file.

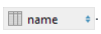
Using the header row. In the Result pane, you can use the cells in the header row (i.e. the row where column names are shown) for:

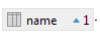
– Sorting data

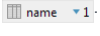
– Reordering columns

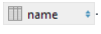
– Hiding and showing columns

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this:  . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance:  . The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order:  .

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled:  .

Here is an example of a table where data are sorted by two of its columns.

	id	name 2	relation 1
1	3	Dylan	brother
2	6	Jack	brother
3	1	Harry	father
4	2	Chloe	mother
5	5	Alice	sister
6	4	Emily	sister

To restore the initial "unsorted" state for the table, click and select Reset View.

To reorder columns, use drag-and-drop for the corresponding cells in the header row. To restore the initial order of columns, click and select Reset View.

	member_id	relation	name 1
1	5	sister	lice
2	1	mother	hloe
3	3	brother	ylan
4	4	sister	mily
5	2	father	arry

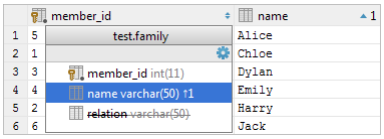
To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

Do one of the following:

1. Do one of the following:
   - Right-click any of the cells in the header row and select Column List.
   - Press `Ctrl+F12`.

   In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

To show all the columns, click and select Reset View.

See also, Using the Structure view to sort data, and hide and show columns.

Parameters pane

The Parameters pane shows the parameters detected in the input pane and lets you edit their values. To open or close this pane, use on the toolbar.

To start editing a value, switch to the corresponding table cell and start typing. To indicate that you have finished editing a value, press `Enter` or switch to a different cell.
To quit the editing mode and restore an initial value, press `Escape`.

When you select a row in the table, the corresponding parameter is highlighted in the input pane.

See also, Executing parameterized statements.

Database Tool Window

View | Tool Windows | Database

On this page:

– Overview of the tool window
– Toolbar icons, context menu commands and shortcuts
– View options
– Icons for data sources and their elements

See also, Working with the Database Tool Window .

Overview of the tool window

The Database tool window provides access to functions for working with databases and DDL data sources . It lets you view and modify data structures in your databases, and perform other associated tasks.

The available data sources are shown as a tree of data sources, schemas, tables and columns. If no data sources are currently defined, use the New command (`Alt+Insert`) to create a data source.

Most of the functions in this window are accessed by means of the toolbar icons or context menu commands. (If the toolbar is not currently shown, click on the title bar and select Show Toolbar.) Many of the commands have keyboard shortcuts. If the toolbar is hidden, the Synchronize and Open Console commands can be access by means of the title bar icons (and respectively).

on the title bar provides access to view options.

Toolbar icons, context menu commands and shortcuts

Icon	Command	Shortcut	Description	Available for
			Use this icon on the title bar to collapse all the nodes.	All node types
	New	`Alt+Insert`	Use this icon, command or shortcut to create a new data source, database console, table, column, index, or a primary or foreign key. The list of options depends on which element is currently selected. See also, Creating a data source, Data Sources and Drivers dialog and Creating a table, a column, an index, or a primary or foreign key.	DB data sources and their elements. If a DDL data source is selected, you can only choose to create another data source.
	Duplicate	`Ctrl+D`	Use this icon, command or shortcut to create a copy of the selected data source. Specify the properties of the data source in the Data Sources and Drivers dialog that opens. See Data Sources and Drivers dialog .	DB and DDL data source nodes
	Synchronize	`Ctrl+Alt+Y`	Use this icon, command or shortcut to update the view of the selected element (i.e. to synchronize the view of the element with its actual state in the database). See also, Auto sync .	DB data sources and their elements
	Properties		Use this icon or command to open the Data Sources and Drivers dialog to manage your data sources and their settings.	All node types
	Disconnect	`Ctrl+F2`	Use this icon, command or shortcut to close the database connection for the selected DB data source or data sources. (The names of the data sources with active database connections are shown in bold.)	DB data sources with active connections and their elements
	Table Editor	`F4`	Use this icon, command or shortcut to open the selected database table in the Table Editor . See also, Working with the Table Editor .	Tables and table columns in DB data sources
	View Editor, Routine Editor, Package Editor	`F4`	Use this icon, command or shortcut to open the definition of the selected view, function, procedure or package in the editor.	Corresponding elements in DB data sources
	Edit Source	`F4`	Use this icon, command or shortcut to open the associated DDL file in the editor.	Tables and table columns in DDL data sources
	Open Console	`Ctrl+Shift+F10`	Use this icon, command or shortcut to open the default database console for the corresponding DB data source.	DB data sources and their elements (tables and table columns)
	Drop Associated		Use the Drop Associated | Primary Key command to remove primary key constraints.	Tables and columns in DB data sources
	Rename	`Shift+F6`	Use this command or shortcut to rename the selected data source, table or table column. Specify the new name in the dialog that opens. See also, Renaming a table or column.	All node types
	Modify Table, Modify Column,	`Ctrl+F6`	Use this command or shortcut to edit the definition of the selected table, column, index or foreign key. See also, Modifying the definition of a table, column, index or a foreign key.	Corresponding elements in DB data sources

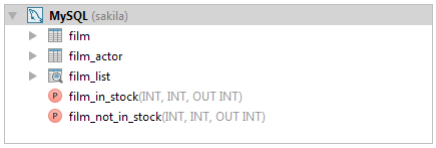
	Modify Index, Modify Foreign Key			
	Copy Reference	`Ctrl+Shift+Alt+C`	Use this command or shortcut to copy the fully qualified name of the selected data source, table or table column to the clipboard.	All node types
	Drop or Remove	`Delete`	Use this command or shortcut to remove the selected item.	All node types
	Find Usages	`Alt+F7`	Use this command or shortcut to find the usages of (references to) the selected item (data source, table or table column) in the project source files and libraries.	All node types
	Compare	`Ctrl+D`	Select two data sources, schemas or tables and then use this command or shortcut to compare table structures for the selected items. The comparison results are shown in the differences viewer .	DB and DDL data sources and tables
	Open New Console		Use this command to create and open a new database console for the corresponding DB data source.	DB data sources and their elements (tables and table columns)
	Forget Schemas		Use this command in problematic cases such as when your data structures start to display incorrectly, fail to synchronize, etc. As a result, IntelliJ IDEA deletes the information it has accumulated about your database. To check if this has eliminated the problem, use the Synchronize command .	DB data sources
	Copy DDL	`Ctrl+Shift+C`	Use this command or shortcut to copy a DDL definition of the selected data source, table or column onto the clipboard.	All node types
	Dump Data to File		Use this command to save the data for the selected table or view in a file, or to view or edit the settings for delimiter-separated values output formats. Select the output format or Configure CSV Formats. In the latter case, the CSV Formats Dialog will open and you'll be able to edit the settings for existing formats, and define new output formats.	Tables in DB data sources
	Color Settings		Use this command if you want to set or change the color for the selected element or elements. (The Database Color Settings dialog will open.)	All node types

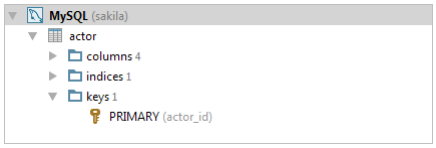
	Diagrams	`Ctrl+Shift+Alt+U`
		
		`Ctrl+Alt+U`
			Use this command or shortcut to see a UML class diagram for the selected data source or table. Select:
			– Show Visualisation to open the diagram on a separate editor tab.
			– Show Visualisation Popup to see the diagram in a pop-up window.
				DB and DDL data sources and tables

		`Ctrl+Q`	Use this key combination to see basic information about the selected element. For example, the info about a table includes the names of the data source, catalog, schema and the table itself, the table definition (`CREATE TABLE`) and, if appropriate, the first 10 rows. To close the documentation pop-up, press `Escape` . You can also access this function from the main menu: View | Quick Documentation.	All node types

View options

The view options, generally, define what is shown in the tool window and how. To view or change these options, click ⚙ on the title bar.

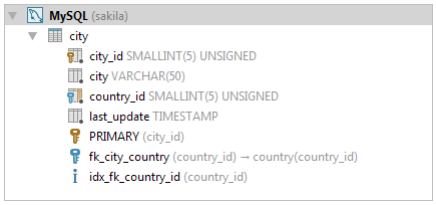
Option	Description
Group Schema	This option defines how schema elements are shown. When on, there are separate nodes for tables, views and stored routines (shown as folders). Tables, views and routines (procedures and functions) are shown as elements of the corresponding groups. When off, there is no explicit grouping for tables, views, and routines. Tables and views are followed by procedures and functions. 
Group Contents	This option defines how table elements are shown. When on, there are separate nodes for columns, indexes, primary and foreign key constraints, and triggers (shown as folders). The elements appear in the corresponding groups.



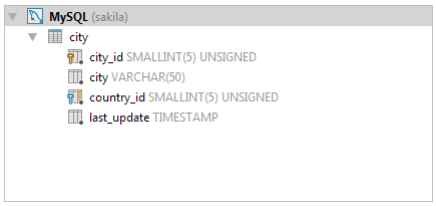
When off, there is no such grouping and, generally, only columns are shown for tables.

Show Keys and etc.

When this option is on, the public and foreign key constraints, and indexes are shown as separate elements.



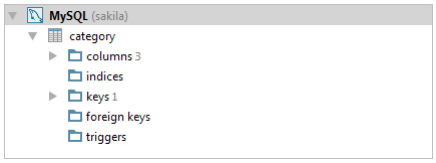
Otherwise, there are no separate elements for the keys and indexes.



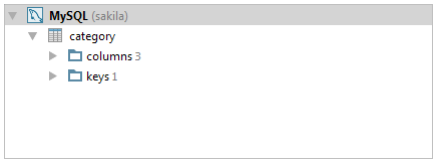
The option is unavailable when the Group Contents option is on.

Show Empty Groups

If the Group Schema or the Group Contents option is on, you can select to show or hide empty groups, i.e. the categories that contain no elements.
The Show Empty Groups option is on:

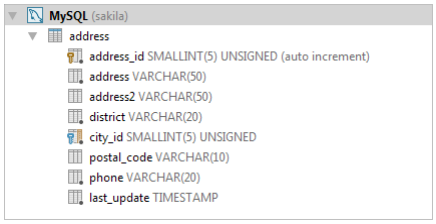


The Show Empty Groups option is off:

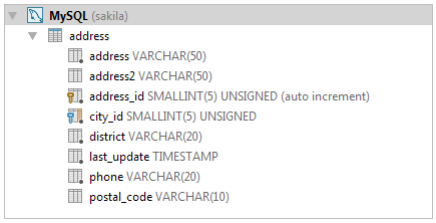


Sort Alphabetically

When this option is off, columns, generally, are unsorted.



When this option is on, the columns are ordered alphabetically.



Show Toolbar

Select or deselect this option to show or hide the toolbar.

The rest of the options are common for all the tool windows, see Viewing Modes .

Icons for data sources and their elements

Icon	Description
	DB data source. Also, DBMS-specific icons are used: DB2

	Derby
	H2
	HSQLDB
	MySQL
	Oracle
	PostgreSQL
	SQL Server
	SQLite
	Sybase
	DB data source with the read-only status, e.g. for DB2.
	DDL data source
	Schema
	Table
	View
	Column
	A `NOT NULL` column
	Column with a primary key
	Column with a foreign key
	Column with an index
	Primary key
	Foreign key
	Index
	Stored procedure or function

Data Sources and Drivers dialog

To access this dialog from the Database tool window : or on the toolbar

Use this dialog to manage your data sources and database drivers, and their settings.

A driver here is understood as a collection that includes database driver files, and also default options and settings for creating a DB data source.

The names of new, yet unsaved items are shown in the left-hand pane in green. New items are saved when clicking Apply or OK.

- Left-hand pane
- Android SQLite data source settings
- DB data source settings
- DDL data source settings
- Driver settings
- Problems

See also, Managing Data Sources .

Left-hand pane

Shown in this pane are your data sources and drivers . When you select an item, its settings are shown in the right-hand part of the dialog.

Use the toolbar icons, context menu commands and associated keyboard shortcuts to manage your data sources and drivers, and also to perform other, related tasks.

Icon and command	Shortcut	Description
Add	Alt+Insert	Use this icon, command or shortcut to create a new data source or driver. Select: – Android SQLite to create a data source for an SQLite database located on an Android device or emulator. For this option to be available, there must be an Android application module in your project and/or an Android SDK has to be defined in IntelliJ IDEA. See Android SQLite data source settings . – The name of DBMS to create a DB data source associated with the corresponding DBMS. If you have created your own drivers, you can also select a driver to use it as the basis for creating the data source. See DB data source settings . – DDL Data Source to create a DDL data source. See DDL data source settings . – Database Driver to create a driver. See Driver settings .
Remove	Delete	Use this icon, command or shortcut to remove the selected item or items from the list.
Duplicate	Ctrl+D	Use this icon, command or shortcut to create a copy of the selected data source or driver.
Go to Driver	Ctrl+B	Use this icon, command or shortcut to view or edit the settings for the driver associated with the selected DB data source.
Make Global or Move to Project		Use this icon or command to move the selected DB data source to the global or project level. (Global data sources are available in all your projects.)
Change Driver		If more than one driver is available for the selected DB data source, use this command to associate the data source with a different driver.
Show Driver Usages	Alt+F7	Use this command or shortcut to find the DB data sources that use the selected driver. The found data sources are shown in the Used By popup which lets you navigate to anyone of the found data sources.
Reset Changes	Ctrl+Alt+Z	Use this command or shortcut to undo the changes made to the selected item.
		Use these icons to switch between the items you've been workings with.

Android SQLite data source settings

Specify the settings for the SQLite database located on an Android device or emulator.

Item	Description
Name	Use this field to edit the name of the data source.
Device	Specify where the target database is located. This may be an Android device connected to your computer or a running Android device emulator. If [none] is the only option in the list, connect the device to your computer or start the emulator. Unless you do that, you won't be able to configure the data source.
Package name	Specify the name of the application package the target database is associated with. For more information about application packages, see Working with AndroidManifest and Android documentation . Select a package name suggested by IntelliJ IDEA or type its ID. For the database to be accessible, the corresponding application must be built as debuggable and installed on the device or the emulator. IntelliJ IDEA run configurations, by default, build Android applications in the debug mode. Alternatively, you can generate the APK in the debug mode .
Storage	Select: – Internal if the database is stored in the internal memory of the device or the emulator. – External if the database is stored in the external memory of the device or the emulator.
Database	If the database is stored in the internal memory, specify the database name. One of the names suggested by IntelliJ IDEA may do. If the database is stored in the external memory, specify the database location relative to the memory root. This may be something like `Android/data/<application_ID>/<database_name>` .
Download	If the necessary SQLite driver files are missing, you can download them by clicking the Download link in the lower part of the dialog.

DB data source settings

- Name
- General tab
- SSH/SSL tab
- Schemas tab
- Options tab for Oracle
- Advanced tab

Name

Item	Description
Name	Use this field to edit the name of the data source.

General tab

Shown on this tab are mainly the database connection settings.

The user interface is adjustable: the set of available controls depends on which option is selected in the list to the right of the URL field .

Item	Description
File	If your database is a local file, specify the path to that file. ... lets you select an existing database file. + lets you create a new database file.
Path	If your database is a local file or folder, specify the path to that file or folder. ... lets you select the database file or folder. Create database. Select this option to create a new database. (This option may be unavailable.)
Host	Specify the hostname (domain name) or the IP address of the computer on which the database is located. If the database is on your local computer, specify `localhost` or `127.0.0.1` . If you are using SSH, the database host must be accessible by the specified domain name or IP address from the computer on which the SSH proxy runs. See SSH/SSL tab .
Port	Specify the database port number.
Database	Specify the name of the target database or schema.
User	Specify the name of the database user (i.e. your database user account name).
Password	Specify the password for the database user .
Save on disk with master password protection	Select this check box if you want IntelliJ IDEA to save the password in its password database. Later, instead of the password, you'll be asked for your master password. (The master password is used to access the password database.)
URL	Shown in this field is the URL that IntelliJ IDEA will use to connect to the database. The user interface for specifying the URL is different depending on which option is selected in the list to the right: – URL only. This option, generally, is for editing the database connection URL directly. When you select this option, only the following fields are available: User , Password and URL. You should edit the URL right in the field. Your user name and password, if necessary, are specified in the corresponding fields, or within the URL in the format appropriate for the JDBC driver that you are using. – When using any other of the options (the options are DBMS-specific), IntelliJ IDEA forms the database connection URL automatically using the info in the fields above the URL field. In all such cases, normally, you don't need to edit the URL (though you can if you want).
Test Connection	Click this button to make sure that the database connection settings are correct and IntelliJ IDEA can communicate with the target database.
Driver	Click the <driver name> link to switch to the settings for the associated driver.
Read-only	Select this check box if you want to protect the data source from accidental data modifications. As a result, you won't be able to modify the data in the Table Editor . Whether data modifications will be possible by means of the consoles depends on the DBMS: IntelliJ IDEA will try to set the database connection status to read-only. All the rest depends on the database driver, i.e. whether and to which extent the driver supports the read-only status. For example, if you define your MySQL data source as read-only, the driver won't let you switch schemas in associated database consoles (see e.g. Selecting the default schema or database). In that case, you should use the Database field to specify your current schema or database.
Auto commit	The default setting for the Auto-commit option in the table editor and the database console .
Auto sync	If this option is off, the view of the data source in the Database tool window is synchronized with the actual state of the database only when you perform the Synchronize command (or `Ctrl+Alt+Y`). If this option is on, the view of the data source is automatically updated: – When you change the data source settings. (Technically, when you click OK in the Data Sources and Drivers dialog.) – When you run DDL SQL statements in the database consoles associated with the data source. Note that auto sync is performed for the overall database and, thus, may be time-consuming. So auto-synchronization is more suitable for "small" databases. If your database is "big", it's recommended that you sync its state manually (e.g. `Ctrl+Alt+Y`), and only for the appropriate database parts such as separate tables.

SSH/SSL tab

If the target database should be accessed using SSH or SSL, select the corresponding check box and specify the associated settings.

SSH

– SSH

– SSL

Item	Description
SSH	
Use SSH tunnel	Select this check box to set up and use an SSH tunnel for accessing a remote database via an SSH proxy.
Copy from	If there is already a data source for which the necessary SSH settings are specified, you can copy those settings from that data source. Click the link and select the data source to copy the settings from.
Proxy host	Specify the hostname (domain name) or IP address of the SSH proxy server that you are using. The SSH proxy server host must be accessible by the specified hostname or IP address from your local computer.
Port	Specify the port on which your SSH proxy server accepts SSH connections. The port number `22` suggested by IntelliJ IDEA is the standard port used by SSH servers. Change this number if your SSH proxy server uses a different port.
Proxy user	Specify the name of the SSH proxy user.
Auth type	Specify the user authentication type used by your SSH proxy. Select: – Password for password-based authentication. – Key pair (Open SSH) for key-based authentication.
Proxy password	For password-based authentication: specify the password for the SSH proxy user. See also, Save on disk .
Private key file	For key-based authentication: specify the path to the file where the corresponding private key is stored. Type the path in the field, or click … (Shift+Enter) and select the file in the dialog that opens .
Passphrase	For key-based authentication: specify the passphrase for the private key if the key is locked with the passphrase.
Save on disk with master password protection	Select this check box if you want IntelliJ IDEA to save the password or the passphrase in its password database. Later, instead of the password or the passphrase, you'll be asked for your master password. (The master password is used to access the password database.)
SSL. All the files specified in this section should be in PEM format. Which of the files you have to specify, depends on the SSL properties of your user account.	
Use SSL	Select the check box to use SSL when connecting to the server.
Copy from	If there is already a data source for which the necessary SSL settings are specified, you can copy those settings from that data source. Click the link and select the data source to copy the settings from.
CA file	Specify the path to SSL Certificate Authority (CA) certificate file. If used, this must be the same certificate as used by the server. Type the path in the field, or click … (Shift+Enter) and select the file in the dialog that opens .
Client certificate file	Specify the path to your (client) public key certificate file. Type the path in the field, or click … (Shift+Enter) and select the file in the dialog that opens .
Client key file	Specify the path to your (client) private key file. Type the path in the field, or click … (Shift+Enter) and select the file in the dialog that opens .

Schemas tab

Specify which schemas and database objects (tables, views, etc.) in the target database you are going to be working with.

You might need to wait a little while IntelliJ IDEA is retrieving the necessary structure information from the database.

Item	Description
Schemas	Shown in this column are the schemas found in the database. Use the check boxes to the left of schema names to select the schemas you want to be working with. Only the selected schemas will be shown in the Database tool window.
Resolve unqualified references	To be able to reference the tables by their "short" names (e.g. in the database console or in your SQL files), select the corresponding check boxes. Otherwise, only fully qualified table names will be treated by IntelliJ IDEA as valid. Note that the DBMS may reject short names when you execute the corresponding statement or statements.
Object filter	To further limit the set of tables and other database objects shown in the Database tool window, specify the corresponding filter. The specified filter is applied to "short" names of the database objects in the selected schemas .
Use legacy introspector	If this check box is present, an older version of the database introspector is available for the corresponding DBMS. (An introcpector is a component that analyzes the database structure.) If there are problems with retrieving the database structure information for your database, you may want to select the check box to use the older introspector version.

Options tab for Oracle

This tab lets you specify which table should be used to store the EXPLAIN PLAN output information.

Item	Description
Explain plan	Selecting None disables the Explain Plan command.
Plan table	The name of the `EXPLAIN PLAN` output table.

Advanced tab

On this tab, you can configure the database connection properties, and also specify the options and environment variables for the database driver JVM.

Item	Description
Name - Value	The set of connection options passed to the database driver as key - value pairs at its start. When you select a cell in the Name column, the description of the corresponding option is shown underneath the table. To find an option of interest, just start typing its name. To start editing a value, click or double-click the corresponding Value field, or press `F2`. To add a row, start editing the values in the last row, where <user defined> and <value> are shown. A new row will be added to the table automatically.
VM Options	The options for the JVM in which the database driver runs. (The driver is started as a separate process in its own JVM.) Example. For certain Oracle Database versions (e.g. for version 9), there may be connection problems when you and your database server are in different time zones. Specifying the time offset for your timezone may help, e.g. `-Duser.timezone=UTC+03:00`. Alternatively, you can try setting the variable `oracle.jdbc.timezoneAsRegion` to `false` in the Name - Value table.
VM Environment	Environment variables for the database driver JVM. Example. Sometimes, when working with Oracle, your data and/or error messages don't display correctly. Many of such problems are encoding-related and can be solved by appropriately setting the `NLS_LANG` variable, e.g. `NLS_LANG=Russian_CIS.CL8MSWIN1251`. For more information, see e.g. Oracle NLS_LANG FAQ. To start editing the variables, click ...

For additional information, refer to your DBMS documentation.

DDL data source settings

A DDL data source is defined by its name, and can reference one or more DDL files and/or another data source (a parent data source).

Item	Description
Name	Use this field to edit the data source name.
DDL Files	Use the controls in this area to compose the list of files that contain the necessary DDL definitions. – + (`Alt+Insert`). Use this icon or shortcut to add a DDL SQL file or files to the data source definition. In the dialog that opens, select the necessary file or files. – – (`Alt+Delete`). Use this icon or shortcut to remove the selected file or files from the list. – ↑ (`Alt+Up`). Use this icon or shortcut to move the selected file one line up in the list. – ↓ (`Alt+Down`). Use this icon or shortcut to move the selected file one line down in the list.
Extend	If necessary, select another data source as a parent. As a result, the data source whose properties you are editing will "inherit" all the DDL definitions from its parent. If the parent data source is not needed, select <none>.

Driver settings

– Name
– Settings tab
– Advanced tab

Name

Item	Description
Name	Use this field to edit the name of the driver.

Settings tab

Shown on this tab are mainly the defaults for the General tab.

Item	Description
Class	The fully qualified name of the driver class to be used.
Dialect	The SQL dialect associated with the corresponding data sources. Via the data sources this dialect will "propagate" to the database console. In addition to particular dialects, also the following option is available: – <Generic>. Basic SQL92-based support is provided including completion and highlighting for SQL keywords, and table and column names. Syntax error highlighting is not available. So all the statements in the input pane are always shown as syntactically correct.
Auto commit	The default setting for the auto commit option.
Auto sync	The default setting for the auto sync option.
URL templates	The templates used to construct the database URL. The text in curly brackets represents variables, e.g. – `{host}` the domain name or IP address of the database host. – `{port}` the database port number. – `{database}` the name of the database or schema. Optional fragments are in square brackets, e.g. `[:{port}]`. Template names correspond to the names of the options in the URL options list.
Driver files	Shown in this area is the list of the driver files. Use: – The download link to download the necessary driver files. – + (`Alt+Insert`) to add the files available locally to the list. – – (`Alt+Delete`) to remove unnecessary files from the list. (No files are physically deleted.)

Advanced tab

Shown on this tab are the default settings for the Advanced tab .

Item	Description
Name - Value	The default set of connection options passed to the database driver as key - value pairs at its start. To start editing a value, double-click the corresponding Value field. To add a row, start editing the values in the last row, where <user defined> and <value> are shown. A new row will be added to the table automatically.
VM Options	The default options for the JVM in which the database driver runs. (The driver is started as a separate process in its own JVM.)

Problems

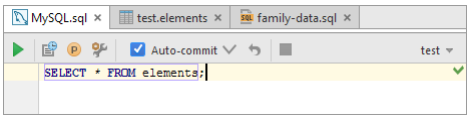
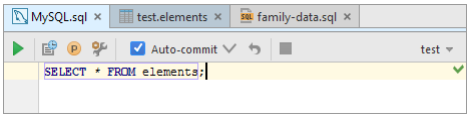
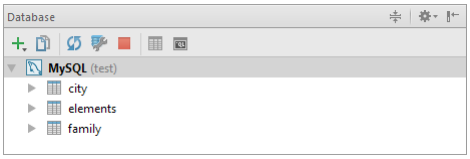
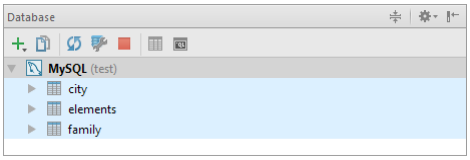
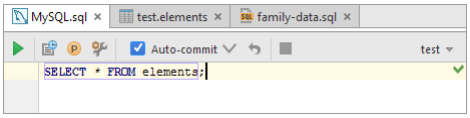
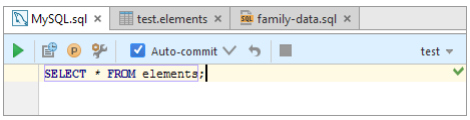
If potential problems are detected, there is a number to the right of Problems. In that case, if you click Problems, you'll see the list of problems as well as controls for fixing them.

Database Color Settings dialog

From the Database tool window :

Select the element or elements of interest, and select Color Settings from the context menu.

Select the color , and specify how this color should be used (see Shared and Set recursively). Use the check boxes in the Appearance Settings section to enable or disable the use of the database colors in various places in the UI.

Item	Description
Color	Click the color that you want to use for the element or elements selected in the Database tool window. If the color that you want is missing, click Custom and specify the color in the dialog that opens. The color which is currently selected is marked with its name, e.g. Blue. If you don't want to use any color, click No Color .
Shared	Select the check box if you want to share the change of color you are about to make with your team. (The database color settings are shared through version control.) In technical terms, the new color (for the selected element or elements) will be stored in – `.idea/databaseColors.xml` or the `.ipr` file if the check box is selected. – `.idea/workspace.xml` or the `.iws` file if the check box is not selected. The `workspace.xml` or the `.iws` file, normally, is not shared through version control while the rest of IntelliJ IDEA configuration files are. For more information on IntelliJ IDEA configuration files and their sharing, see Project .
Set recursively	If the check box is selected, the selected color will be applied to the element itself and all its subordinate elements recursively. If the check box is not selected, the colors set individually for the subordinate elements won't change.
Appearance Settings	
Enable database colors	Clear the check box to (temporarily) disable the use of the database colors everywhere in the UI.
Use in editor tabs	Select the check box to enable the use of the database colors for editor tabs (when working with the input pane of the database console and the table editor). When this option is off, the tabs look like this:  When this option is on, the tabs have the same color as the corresponding data source or table (e.g. blue): 
Use in database view	Select the check box to enable the use of the database colors in the Database tool window. When this option is off, the tool window looks like this:  When this option is on, the tool window may look as shown below. (In this example, the blue color is set for the corresponding data source.) 
Use in toolbars	Select the check box to use the database colors to paint the background of the toolbars in the input pane of the database console and the table editor. When this option is off, the toolbar looks like this:  When this option is on, the toolbar background has the same color as the corresponding data source (e.g. blue). 

No Color	Click this button, if you don't want to use any color for the selected element or elements.

Debug Tool Window

View | Tool Windows | Debug

`Alt+5`

In this topic:

- Overview
- Debug toolbar
- Stepping toolbar
- Hide/restore toolbar
- Moving tabs and areas
- Context menu of a tab

Overview

The Debug tool window becomes available when you start debugging .

It displays the output generated by the debugging session for your application. If you are debugging multiple applications, the output for each application is displayed in a separate tab named after the corresponding run/debug configuration .

For each application, there are the following nested tabs:

- Console : displays system information and error messages, and the console input and output of your application.
- Debugger tab divided into the following areas:
  - Frames / Threads
  - Variables
  - Watches
- Logcat that is only available for Android applications and displays a log of all processes currently running on the connected device or an emulator.
- Dump : opens when the Get thread dump button is clicked on the Debugger toolbar .
- Elements : appears if you are using Chrome browser for debugging.

Each area has a context menu that allows you to configure its behavior and navigate between tabs.

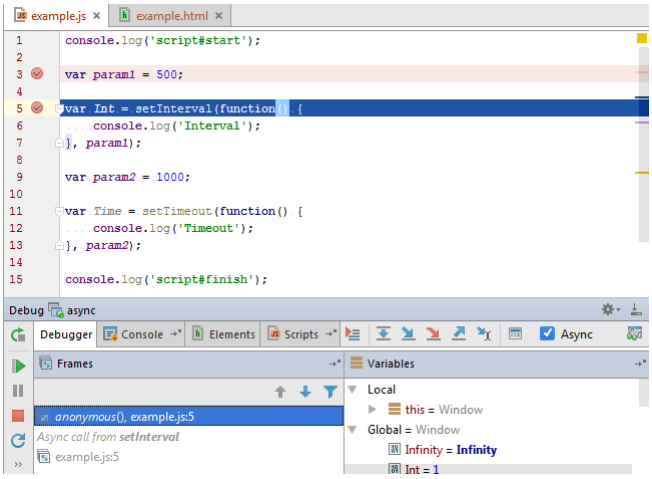
Each of the tabs and areas can be hidden/restored , or moved to a location of your choice.

Debug toolbar

Item	Tooltip and Shortcut	Description
	Rerun `Ctrl+F5`	Click this button to stop the current application and run it again. When an application is stopped, this button toggles to .
	Debug `F9`	When the current application is stopped, click this button to debug it again. When an application is running, this button toggles to .
	Resume Program `F9`	When an application is paused, click this button to resume program execution.
	Pause Program `Ctrl+Pause`	Click this button to pause program execution.
	Stop `Ctrl+F2`	Click this button to terminate the current process externally by means of the standard `shutdown` script.
	View Breakpoints `Ctrl+Shift+F8`	Click this button to open the Breakpoints dialog box where you can configure breakpoints behavior.
	Mute Breakpoints	Use this button to toggle breakpoints status (enabled/disabled). You can temporarily disable all breakpoints in the project to execute the program without stopping at breakpoints.
	Get thread dump	Click this button to open the Dump tab .
	Restore Layout	Click this button abandon changes to the current layout and return to the default state.
	Settings	Click this button to open the menu with the following options available: – Show Values Inline: select this option to enable the Inline Debugging feature that allows viewing the values of variables right next to their usage in the editor. – Show Method Return Values: select this option to display the return values of the last executed method. – Auto-Variables Mode: select this option if you want IntelliJ IDEA debugger to automatically evaluate certain variables (the variables at breakpoints plus several lines before and after the breakpoint). – Sort Values Alphabetically: select this option to sort the values in the Variables pane in the alphabetical order. – Unmute breakpoints on session finish: select this option to re-enable all disabled breakpoints after the debugging session has finished.
	Pin	Click this button to pin or unpin the currently selected tab.

		Close Ctrl+Shift+F4	Click this button to close the selected tab.
		Help F1	Click this button to open the corresponding help page.

Stepping toolbar

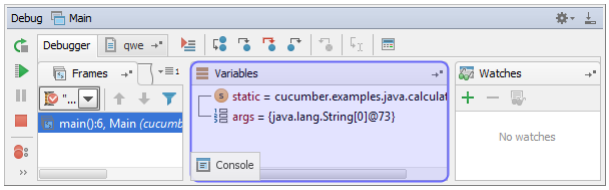
Item	Tooltip and Shortcut	Description
	Show Execution Point Alt+F10	Click this button to highlight the current execution point in the editor and show the corresponding stack frame in the Frames pane.
	Step Over F8	Click this button to execute the program until the next line in the current method or file, skipping the methods referenced at the current execution point (if any). If the current line is the last one in the method, execution steps to the line executed right after this method.
	Step Into F7	Click this button to have the debugger step into the method called at the current execution point. If stepping into the called method is suppressed through the Stepping page of the Settings dialog box (for example, if it is of a standard Java SDK class or a simple getter), the method will be skipped. Change the settings or use the Force Step Into command.
	Force Step Into Shift+Alt+F7	Click this button to have the debugger step into the method called in the current execution point even if this method is to be skipped.
	Step Out Shift+F8	Click this button to have the debugger step out of the current method, to the line executed right after it.
	Drop frame	Interrupts execution and returns to the initial point of method execution. In the process, it drops the current method frames from the stack.
	Run to Cursor Alt+F9	Click this button to resume program execution and pause until the execution point reaches the line at the current cursor location in the editor. No breakpoint is required. Actually, there is a temporary breakpoint set for the current line at the caret, which is removed once program execution is paused. Thus, if the caret is positioned at the line which has already been executed, the program will be just resumed for further execution, because there is no way to roll back to previous breakpoints. This action is especially useful when you have stepped deep into the methods sequence and need to step out of several methods at once. If there are breakpoints set for the lines that should be executed before bringing you to the specified line, the debugger will pause at the first encountered breakpoint. Tip Use this action when you need a kind of a temporary breakpoint at a specific line, where program execution should not be interrupted.
	Evaluate Expression Alt+F8	Click this button to open the Evaluate Expression dialog.
Async		– When this check box is selected, IntelliJ IDEA recognizes breakpoints inside asynchronous code and stops at them and lets you step into asynchronous code. As soon as a breakpoint inside an asynchronous function is hit or you step into asynchronous code, a new element `Async call from <caller>` is added in the Frames pane of the Debugger tab. IntelliJ IDEA displays a full call stack, including the caller and the entire way to the beginning of the asynchronous actions. – When the check box is cleared, IntelliJ IDEA does not recognize and therefore skips breakpoints in the asynchronous code and does not allow you to step into it. The image below shows an example of a JavaScript debugging session.  – With the Async check box selected, the debugger will stop at line3(breakpont), then at line5(breakpoint). On clicking Step into, the debugger will stop at line5 (on `function`), then will move to line6. – With the Async check box cleared, the debugger will stop at line3(breakpont), then at line5(breakpoint). On clicking Step into, the debugger will move to line9.

Hide/restore toolbar

Icon	Tooltip	Description
	Hide	Click this button located in the upper-right corner of the Debug Console, Watches, Treads, Frames, or Variables pane to hide the corresponding area. When an area is hidden, its icon appears in upper-right corner of the Debugger.
	Restore 'Console' view	Click this button to make the Console area visible. This button becomes available after clicking .
	Restore 'Frames' view	Click this button to make the Frames area visible. This button becomes available after clicking .
	Restore 'Watches' view	Click this button to make the Watches area visible. This button becomes available after clicking .
	Restore 'Threads' view	Click this button to make the Threads area visible. This button becomes available after clicking .
	Restore 'Variables' view	Click this button to make the Variables area visible. This button becomes available after clicking .

Moving tabs and areas

If you are unhappy with the default layout of the Debug tool window, you can always move the tabs and areas. To to that, just drag a tab or an area to the desired location. The possible target gets highlighted:



Drop the tab or area in the highlighted location.

To restore the default layout of tabs and area, click in the Debug toolbar.

Context menu of a tab

Use the context menu of the Frames / Threads , Variables or Watches areas to configure the behavior of these areas or navigate between tabs.

Item	Description
Hide	Click this button to hide the corresponding area
Close Others	Click this button to hide all tabs except for the Console and Debugger tabs.
Focus On Startup	If this option is selected, the selected area gets the focus when you start a debugging session.
Focus On Breakpoint	If this option is selected, the selected area gets the focus when a breakpoint is reached.
Select Next Tab / Select Previous Tab	Use these options to switch between the Console and the Debugger tabs.

`Ctrl+Alt+Right` / `Ctrl+Alt+Left`

Debug Tool Window. Debugger

The Debugger tab is divided into the following areas:

- Frames / Threads
- Variables
- Watches

Debug Tool Window. Frames

The Frames pane enables you to gain access to the list of threads of your application. You can also export to a text file and customize thread presentation . To examine a thread, select it from the drop-down list on top of the pane. The status and type of a thread is indicated by a special icon and a textual note next to the thread's name. For each thread, you can view the stack frame, examine frames, navigate between frames, and automatically jump to a frame's source code in the editor.

To examine the values stored in a frame, use the Variables pane of the Debug tool window.

In this topic:

– Thread Status
– Threads Icons
– Context menu options
– Toolbar

Thread Status

Thread status	Description
RUNNING	The thread is active and running.
WAIT	The thread is waiting for a monitor.
UNKNOWN	The status of the thread cannot be determined.

Threads Icons

Icon	Description
	A thread group, or a collection of related threads that can be managed as a unit.
	The current thread group.
	An active thread.
	A suspended thread.
	A frozen thread.
	A thread at breakpoint.
	The current thread at breakpoint.

Context menu options

Item	Description
Drop Frame	Use this command to drop the selected frame , i.e. go back in time while debugging. This option is only available if there are two or more frames.
Force Return	Use this command to force a return from the current method before the return statement is reached and without executing any more instructions from it. If the method returns a value, you will be prompted to specify it (smart code completion is provided). If the method has try-finally blocks, you will be prompted to choose whether you want to execute them or not.
Add Stepping Filter	Use this command to add a stepping filter in the dialog that opens.
Export Threads	Use this item to open the Export Threads dialog box that allows you to export a thread to the specified text file.
Customize Threads View	Use this item to manage contents of the Frames tab. For example, you can opt to show thread groups, line numbers etc. Refer to the dialog description for details.

Toolbar

Item	Shortcut and Tooltip	Description
	Previous Frame/Next Frame `Ctrl+Alt+Up` / `Ctrl+Alt+Down`	Use the arrow buttons to navigate through the frame stack.
	Hide Frames from Libraries	Click this button to hide frames from libraries. If this button is released, all frames are displayed.

Debug Tool Window. Threads

The Threads pane shows all threads of a process as a tree view, and allows exploring them, customizing thread view , and exporting to a file .

Thread Status

Thread status	Description
RUNNING	The thread is active and running.
WAIT	The thread is waiting for a monitor.
UNKNOWN	The status of the thread cannot be determined.

Threads Icons

Icon	Description
	A thread group, or a collection of related threads that can be managed as a unit.
	The current thread group.
	An active thread.
	A suspended thread.
	A frozen thread.
	A thread at breakpoint.
	The current thread at breakpoint.

Context menu options

Item	Description
Suspend	Select this option to suspend the selected thread. When a thread is suspended, this menu option toggles to Resume.
Interrupt	Use this command to interrupt the active thread.
Drop Frame	Use this command to drop the selected frame , i.e. go back in time while debugging. This option is only available if there are two or more frames.
Add Stepping Filter	Use this command to add a stepping filter in the dialog that opens.
Export Threads	Use this item to open the Export Threads dialog box that allows you to export a thread to the specified text file.
Customize Threads View	Use this item to manage contents of the Frames tab. For example, you can opt to show thread groups, line numbers etc. Refer to the dialog description for details.

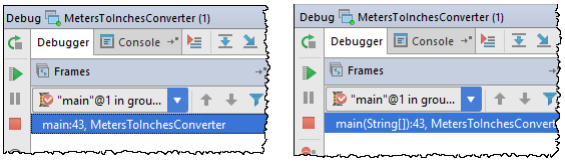
Toolbar

Item	Shortcut and Tooltip	Description
	Previous Frame/Next Frame `Ctrl+Alt+Up` / `Ctrl+Alt+Down`	Use the arrow buttons to navigate through the frame stack.
	Hide Frames from Libraries	Click this button to hide frames from libraries. If this button is released, all frames are displayed.

Customize Threads View

Debug tool window | Frames | context menu | Customize Threads View Debug tool window | Threads | context menu | Customize Threads View

Use this dialog box to manage contents view and groupings of the Frames tab.

Item	Description
Show thread groups	If this option is selected, threads appear in a hierarchy of thread groups, allowing you to drill down into each group to access its member threads.
Show stack frames for synthetic methods	Specify whether you want stack frames for synthetic methods to be shown.
Move current thread to the top	Select this check box to keep the current thread on top of the list.
Show line number	If this check box is selected, the line number is displayed.
Show class name	If this check box is selected, the name of a class containing the method is shown.
Show package name	If this check box is selected, the name of the class package is shown. This check box allows hiding package names in the Threads view.
Show source file name	Shows the name of source file that contains the method.
Show method arguments types	If this check box is selected, the method type is shown next to the method name in the Frames or Threads panes of the Debugger. For example, the unselected vs selected check box is shown in the following image: 

Export Threads

Run | Export Thread

Debug tool window | Threads | context menu | Export Threads

Alt+U+H

Use this dialog to export a thread to the specified text file.

Item	Description
Export to file	Type the target file to store the thread report.
...	Use this button to navigate to the desired location in the Select Path dialog.
Save	Click to save the current threads in a text file indicated in the Export to file field.
Copy	Click to copy the thread status to the Clipboard.
Cancel	Use to discard all changes and close the dialog.

Debug Tool Window. Variables

In this topic:

- Overview
- Toolbar
- Context menu
- Variable types

Overview

The Variables pane enables you to examine the values stored in the objects of your application.

When a stack frame is selected in the Frames pane , the Variables pane displays all data within its scope (method parameters, local and instance variables).

In this pane, you can set labels for objects, inspect objects, evaluate expressions, add variables to watches and more.

Toolbar

This toolbar appears only on clicking the button in the Watches pane . So doing, the focus is with the Debugger tab .

Item	Shortcut	Description
	`Alt+Insert`	Click this button to create a new watch.
	`Alt+Delete`	Click this button to remove the selected watch from the list.
	`Alt+Up` `Alt+Down`	Use these buttons to change the order of watches.
	`Ctrl+D`	Use this button to create a copy of the selected watch.
	N/A	Use this button to show watches in the Variables pane . When this button is pressed, the corresponding Variables pane shows this toolbar and the created watches.

Context menu

Item	Shortcut	Description
Inspect	N/A	This command is available for fields, local variables and reference expressions, and opens a non-modal Inspection window, where you can concentrate on a particular reference. You can open as many Inspection windows as required. The view in the Inspection window is the same as in the Watches pane, but requires less screen space.
Mark Object	`F11`	Use this command to add an object label.
Set Value	`F2`	Use this command to change the runtime value of a field or a variable.
Copy Value	`Ctrl+C`	Use this command to copy the value of the selected variable to the Clipboard. If multiple items are selected, not only variables' values, but also their structure is copied, so that when you copy-paste the selection to a text file, the indentation mimics the tree output of the debugger to produce an easy-to-read output. Alternatively, hover your mouse cursor over a value and view its contents in the tooltip.
Copy Value As	N/A	This menu item is available only in the PHP context. Choose this command to copy the selected variable to the Clipboard in one of the following formats: – print_r – var_export – json_encode
Copy JSON		This menu item is available only in the JavaScript context. Choose this command to copy the selected value in the JSON format .
Compare Value with Clipboard	N/A	Use this command to compare the selected value with the value currently in the Clipboard.
Copy Name	N/A	Use this command to copy the name of the selected variable to the Clipboard.
Evaluate Expression	`Alt+F8`	Use this command to evaluate the selected variable in the dialog that opens.
Add to Watches	N/A	This command is available for all nodes except static ones. Use this command to create an expression that references the node and add this expression to the Watches pane.
Show Referring Objects	N/A	Use this command to display a list of objects referring to the currently selected variable.
Jump to Source	`F4`	This command opens the source code of the selected variable or field in the editor and places the caret in the corresponding line.
Jump to Type Source	`Shift+F4`	Use this command to open the source code of the selected variable or field type in the editor.
View Text / View/Edit Text (for string values)	N/A	Use this command to display the text representation (if any) of the selected variable. Note that for the string values View Text turns into View/Edit Text. This makes it possible to change string values in a text area.
View as	N/A	Use this command to select how you want variable values to be displayed. For numeric values, you can toggle between the decimal and hexadecimal formats. Objects are usually represented by their class name and instance identifier, but you can optionally show them in the string format. – Auto: this option is available for items where different layout is possible. If selected, IntelliJ IDEA looks through all available renderers and searches for the first suitable layout for the currently selected item. If none is found, the default layout is applied. – Hex: this option is available for numeric variables. If selected, the variable is shown in the hexadecimal format. – Primitive: this option is available for primitive variables. It shows the value appropriate for the corresponding primitive type. – Binary: this option is available for binary data. If selected, the variable is shown as a binary literal. – Timestamp: this option is available for `long` data. If selected, the variable is shown as a timestamp. – Boolean: available for boolean data. If selected, the variable is shown as a boolean value (`true` or `false`). – toString(): this option is available for all reference types where `toString()` is overridden (except for arrays), and shows the node's `toString()` value in the tree. – Array: this option is available for arrays. – Object: this is the default layout available for all non-primitive variable types.

		– Map: this options shows as a map. – Collection: this option shows as a collection. – File: this option is available for files and is the default layout for `java.io.File` type variables. – User-defined renderer: this option is available for renderers created by the user in Debugger | Type Renderers , or in the Customize Data Views dialog. The corresponding renderer name is shown in the menu.
Adjust Range	N/A	This command is available for arrays and lists and lets you view the contents of an array within the specified range of indices.
Show types	N/A	Select this option if you want to show variables' types.
Customize Data Views	N/A	Select this option to open the Customize Data Views dialog where you can configure how data is represented.

Variable types

The icon on the left of each variable indicates its type:

– : static
– : global
– : field
– : array
– : primitive
– : object

Customize Data Views

Debug tool window | Variables tab | context menu | Customize Data Views

Use this dialog to configure the debugger data view options. Note that the same settings can be alternatively configured on the Debugger page in the Settings dialog.

- Java tab
- Java Data Type Renderers tab

Java tab

Item	Description
Autoscroll to new local variables	Select this option to automatically scroll to new variables that appear in the scope when stepping.

Show value tooltip	Select this option to enable automatic display of tooltips for values.
	A tooltip in this context is a pop-up that provides an alternative, sometimes a more convenient presentation of values in the Variables pane of the Debug tool window.
	
	To illustrate, let's assume that there is a statement like this in your code:
	
	`String s = "Hello, World!\n Hello, World!";`
	
	When this statement is executed in the debugger, you'll see a line looking similar to this in the Variables pane:
	
	`s = {java.lang.String@62} "Hello, World!\n Hello, World!"`
	
	with the line break shown as `\n`.
	
	If the Show value tooltip option is on and you click this line and then hold the mouse pointer on it, you'll see a yellow area (the "tooltip") in which the value of `s` is shown as
	
	`Hello, World!`
	
	`Hello, World!`
	
	with a real line break in place of `\n`.
	
	If this option is disabled, press `Alt` to display a value.

Show	In this section, select which elements you want the Debugger to display: – Declared type – Synthetic fields – $val fields as local variables – Fully qualified names – Object id – Static fields – Static final fields
Show hex value for primitives	Select this option if you want numeric variables to be displayed in the hexadecimal format.
Hide null array elements	Select this option if you want null array elements to be omitted.
Enable alternative view for Collection classes	Select this option to display collections and maps in a more convenient format.
Enable `toString()` object view	In this section, you can select classes if you need them and their descendants to be presented as a result of the `toString()` method call while debugging. Use the following controls: – For all classes that override toString() method: select this option to show all classes as `toString()`. – For classes from the list: populate the list of classes to be shown as `toString()`, using the , , and the buttons. Use the check boxes next to the class names to temporarily enable or disable particular filters. – : click this button to add a class to the list using the Choose Class dialog. – : click this button to add a custom class filter using the New Filter dialog. To define a filter, enter a string pattern, e.g. `*.Test`, `javax.swing.*`, etc. – : click this button to remove a filter from the list.

Java Data Type Renderers tab

On this tab, you can create and customize rendering schemes for data presentation in the debugger Frame view.

If no rendering scheme is currently defined, start by clicking +.

Item	Description
+	Use this icon to add a new rendering scheme to the list.
–	Use this icon to remove the selected scheme from the list.

	Use this icon to create a copy of the selected scheme.
	Use these icons to move the selected item one line up or down in the list. Note that the order of items in the list determines which renderer is used in the case of ambiguity born of class inheritance.
Renderer name	In this field, specify the name of a new renderer, or edit an existing renderer name.
Apply renderer to objects of type (fully-qualified name)	In this field, specify the type of objects that will be represented with this renderer. Type a fully qualified object name, or click the Browse button and choose the desired type from the list in the Renderer Reference Type dialog.
When rendering a node	This option determines how an object is displayed in the debugger when nodes are collapsed: – Use default renderer: select this option to display the node in the default way. – Use following expression: enter the java expression you want to use to identify an object. You can use object properties, constants, and even a string math as part of your renderer. Note that you can use code completion (`Ctrl+Space`) when defining expressions. All method calls and member variable access are relative to the object you're rendering. Use `this` to refer to an instance to which the renderer applies. **Warning!** – Using heavy expressions in renderers may slow down data rendering in views. – Method calls should be used with caution because of possible side-effects.
When expanding a node	This option determines how an object is displayed in the debugger when nodes are expanded: Normally, expanding a node in the debugger lists the object's member variables (using the renderer appropriate for the corresponding object types). This option lets you override this behavior and select a single expression or a series of expressions to be displayed. You may use this to limit the amount of information displayed, or to be more precise in how the information is presented. – Use default renderer: select this option to display the node children in the default way. – Use following expression: enter the Java expression you want to use to identify an object. Test if a node can be expanded (optional): enter a Boolean expression. If it is `true` , the renderer displays expandable nodes for the defined objects. Otherwise, no nodes are displayed. – Use list of expressions: create a list of separate expressions to be calculated and presented as node children. Use: + (`Alt+Insert`) to create a new expression. – (`Alt+Delete`) to remove the selected expression from the list. ↑ (`Alt+Up`) to move the selected expression one line up in the list. ↓ (`Alt+Down`) to move the selected expression one line down in the list. Note that you can use code completion (`Ctrl+Space`) when defining expressions.

Debug Tool Window. Watches

In this topic:

- Overview
- Toolbar
- Context menu

Overview

In the Watches pane you can evaluate any number of variables or expressions in the context of the current stack frame. The values are updated with each step through the application, and become visible every time the application is suspended.

While the Evaluate Expression command on the context menu of the Variables pane enables you to see one expression at a time, the Watches pane shows multiple expressions that persist from one debug session to another, until you remove them.

You can create watches in this pane, in the Variables pane and even in the editor.

Watch expressions are always evaluated in the context of a stack frame that is currently inspected in the Frames pane. If an expression cannot be evaluated, it is displayed with a question mark.

Toolbar

Item	Shortcut	Description
	`Alt+Insert`	Click this button to create a new watch.
	`Alt+Delete`	Click this button to remove the selected watch from the list.
	`Alt+Up` `Alt+Down`	Use these buttons to change the order of watches.
	`Ctrl+D`	Use this button to create a copy of the selected watch.
	N/A	Use this button to show watches in the Variables pane . When this button is pressed, the corresponding Variables pane shows this toolbar and the created watches.

Context menu

Item	Shortcut	Description
New Watch	`Insert`	Choose this command to create a new watch. A text field opens, where you can enter new watch expression.
Remove Watch	`Delete`	Choose this command to delete the currently selected watch expression from the list.
Edit	`F2`	Choose this command to change the selected watch expression.
Remove All Watches		Choose this command to delete all watch expressions from the list.
Inspect		Available for fields, local variables and reference expressions. Choose this command to open the Inspect window for the node, which allows you to perform the same operations as those available in the stack frame, with the only difference that the root node is the one you have selected. You can recursively call the new Inspect windows from within each other. Each window is not modal and immediately reflects all changes in its subtree.
Mark Object	`F11`	Choose this command to add a label to an object.
Set Value		This command is available on the context menu of a field or local variables, added to a watch and being currently in scope. Choose this command to assign the desired value to such variable.
Show Referring Object		Choose this command to display the list of objects referring to the current watch.
Jump to Source	`F4`	This command opens the source code of the selected variable or field in the editor and places the caret on a proper line.
Jump to Type Source	`Shift+F4`	This command opens the source code of the type of selected variable or field in the editor.
View as		Use this command to select the layout of the values. For the numeric values, you can toggle between the decimal and hexadecimal formats. The objects are usually represented by their class name and instance identifier, but you can optionally show them in string format. – Auto: Available for items where different layout is possible. With this layout IntelliJ IDEA looks through all available renderers searching for the first suitable layout for the current item. If none is found, the default layout is applied. – Hex: This layout is available for numeric variables. If checked, the variable is shown in hexadecimal format. – Primitive: This layout is available for primitive type variables. It shows the value appropriate for the primitive type. – toString(): This layout is available for all reference types where `toString()` is overridden except for arrays, and shows the node's `toString()` value in the tree. – Array: Available for arrays. – Object: Default layout. Available for all non-primitive type nodes. – Map: Show as a map. – Collection: Show as a collection. – User-defined renderer: Available for renderers created by the user in the Debugger | Type Renderers or in the Customize Data Views dialogs. The corresponding renderer name is shown in the menu.
Copy Value	`Ctrl+C`	Copies the selected node value to the clipboard.
Copy Name		Copies the selected node name to the clipboard.

Debug Tool Window. Console

This tab is marked with and shows the output and error stream messages.

On this page:

– Console Toolbar
– Context Menu Options
– Keyboard Shortcuts

Console Toolbar

Item	Tooltip and shortcut	Description

	Up/down the Stack Trace
	`Ctrl+Alt+Up`
	`Ctrl+Alt+Down`
		Click this button to navigate up or down in the stack trace and have the cursor jump to the corresponding location in the source code.

	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location in the source code.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.
	Toggle tasks executions/text mode	Click this button to view Gradle task execution in a tree mode.

Context Menu Options

Item	Description
Compare with Clipboard	Opens the Clipboard vs Editor dialog box that allows you to view the differences between the selection from the editor and the current clipboard content. This dialog is a regular comparing tool that enables you to copy the line at caret to the clipboard, find text, navigate between differences and manage white spaces.
Fold Lines Like This	Opens the Console dialog that allows you defining the lines to be folded to hide extraneous information.
Copy URL	Choose this command to copy the current URL to the system clipboard. This command only shows on a URL, if it is included in an application's output.
Create Gist	Choose this command to open the Create Gist dialog box.
Clear All	Clears the output window.

Keyboard Shortcuts

The `Ctrl+D` key combination allows you to send EOF (end of file), i.e. to signal that no more data can be read from a data source.

Debug Tool Window. Dump

This tab appears when the Dump button is clicked on the Debugger toolbar , or on choosing Run | Get Thread Dump on the main menu.

Use this tab to review the external thread dump or the one taken from the debugger in a handy way. The tab is divided into two parts. The left one displays all threads, and the right one - the stack trace for the selected thread.

In the Dump tab, all threads are sorted so that the most meaningful and useful threads are on top of the list. For your convenience, threads are displayed in varying shades of gray text, and deadlocks are highlighted in red.

In this topic:

– Threads toolbar
– Thread types

Threads toolbar

Item	Tooltip and Shortcut	Description
	Filter `Ctrl+F`	Click this button to enable filtering the thread dump by a word in the stack trace. A search field appears in the list of threads, where you can type the search string. Note that plain search `Ctrl+F` is also available in the right-hand part of the Dump tab, in the stack trace for the selected thread.
	Copy to Clipboard	Click this button to copy the whole thread dump to the Clipboard.
	Sort threads by name / Sort threads by type	Click these buttons to toggle between sorting threads in the alphabetical order, or by type.
	Export To Text File `Alt+O`	Click this button to export the current threads to a specified text file.

Thread types

Item	Description
	Thread is suspended.
	Thread is waiting on a monitor lock.
	Thread is running.
	Thread is executing network operation, and is waiting for data to be passed.
	Thread is idle.
	Event Dispatch Thread that is busy.
	Thread is executing disk operation.

Debug Tool Window: Elements Tab

In this tab, view the HTML source code that implements the active browser page and its HTML DOM structure . Any changes made to the page through the browser are immediately reflected in the tab.

To Have the Tab Displayed:

1. Make sure you are using the Chrome browser for debugging.
2. Make sure the JetBrains Chrome Extension is installed in your Chrome browser.
   The extension is installed when you first initiate a debugging session and follow the dedicated link provided by IntelliJ IDEA. For details, see Using JetBrains Chrome Extension .
3. Make sure the LiveEdit repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The Structure, Text, and Scripts Panes

The tab consists of three panes: Structure, Text, and Scripts.

The Structure pane shows the HTML DOM structure of the page that is currently active in the browser. The structure is updated dynamically according to the changes made on the page.

The Text pane shows HTML source code of the page that is currently active in th browser. The code is updated dynamically according to the changes made on the page.

The Scripts pane shows a tree of executed scripts.

The Structure and Text panes are mutually synchronized. When you click a node in the DOM structure, IntelliJ IDEA scrolls through the contents of the Text pane.

The panes are also synchronized with the browser. IntelliJ IDEA highlights the element in the browser as soon as you click the corresponding node in the DOM structure or in the Text pane.

Debug Tool Window. Logcat

The Logcat tab is only available for Android applications (see Debugging with Logcat for details). It displays a log of all processes currently running on the connected device or the emulator.

The logcat functionality is also available from the Android tool window . The only difference is that while the Logcat tab in the Debug tool window shows a log of all processes, in the Android tool window you can filter log messages by a specific process.

Toolbar

Use the toolbar controls and buttons to configure the scope and the presentation of log data, and to navigate through the log.

Item	Tooltip and shortcut	Description
Log Level	N/A	From this drop-down list, select the priority of log messages to be displayed. The available options are: – Verbose – Debug – Info – Warn – Error – Assert
	Find	Use this text box to search through the list of messages. As you type a search string, the messages that match the search pattern are displayed with the matching character strings highlighted. To finalize the search, press `Enter`. Search patterns are stored in the search history list. To clear the search history, click the button.

Filters	N/A	From this list, select an existing filter configuration, or create a new one.
		A filter configuration is a set of filtering parameters. The use of filter configurations provides more flexible control over the type and amount of log data displayed than just specifying the information type by choosing a message priority in the Log level drop-down list.
		– To display messages related to the selected process, select Show only selected application.
		– If you want full log data to be displayed, select No Filters from the drop-down list.
		– To apply a filter, select it from the drop-down list.
		– To create a new filter configuration, select Edit Filter Configuration. In the Create New Logcat Filter dialog box that opens, click the Add toolbar button + and specify the following filtering parameters:
		  – Log Tag: use this parameter if you only want messages from a certain system component to be displayed. Type a regular expression to specify the tag that indicates the relevant system component, such as ActivityManager, AudioService, etc., or a user-defined tag. For more details, see Filtering Log Output .
		  – Log Message: use this parameter if you only want messages that contain certain elements or character strings to be displayed. Type a regular expression that defines the character string to be detected.
		  – Package Name: use this parameter if you only want messages that refer to a specific Java package (class path) to be displayed.
		  – PID: use this parameter if you only want messages that refer to a specific process (process ID) to be displayed.
		  – Log Level: use this parameter if you only want messages with a certain priority level to be displayed.
		– To update a filter, select Edit Filter Configuration, then in the Create New Logcat Filter dialog that opens select a filter from the list and edit the filter values.
		– To remove a filter configuration from the list, choose Edit Filter Configuration. In the Create New Logcat Filter dialog box that opens, select the filter and click the Delete toolbar button –.

	Up the Stack Trace `Ctrl+Alt+Up`	Click this button to navigate up in the stack trace and have the cursor jump to the corresponding location in the source code.
	Down the Stack Trace `Ctrl+Alt+Down`	Click this button to navigate down in the stack trace and have the cursor jump to the corresponding location in the source code.
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to move the caret to the last line of the console output.
	Print	Click this button to print the logs.
	Clear logcat	Click this button to remove log data from previous sessions on the selected device.
	Restart	Click this button to restart logging.

Dependency Viewer

View | Tool Windows | Dependency Viewer

The Dependency Viewer tool window displays results of analyzing dependencies, backward and cyclic dependencies.

In this section:

- Panes of the Dependency Viewer
- Analyzed Code Toolbar
- Usage Toolbar and Context Menu

Panes of the dependency Viewer

The Dependency Viewer consists of the following panes:

- Analyzed Code pane in the upper-left part of the tool window containing a tree view of your project's files and packages. Selecting a node (package or file), for which you want to find dependencies, populates the Parent Code pane.
- Parent Code pane in the upper-right part of the tool window represents the classes your selection depends on. In addition to the other classes in project, these dependencies also include any classes in the libraries and test sources, if the corresponding view filter is enabled.
- Usage pane in the lower-left part of the tool window is populated when you select an entry in the Parent Code pane.
  - For the dependencies analysis this pane contains the objects which your code is dependent on (that is, the code in the left pane uses them).
  - For the backward dependencies analysis , these are the objects that depend on your code (that is, they use something of the code in the left pane).
  - For the cyclic dependencies analysis this pane shows the objects which the analyzed code refers to and which, in turn, refer back to your code.

Analyzed Code Toolbar

Item	Tooltip and Shortcut	Description
	Close	Click this button to close the current tab of the tool window.
	Rerun Ctrl+F5	Click this button to rerun the dependency analysis in the same tab.
	Flatten Packages	When the button is pressed, all packages display as a single-level tree view.
	Show Files	When the button is pressed, files display in the Analyzed Code and Parent Code panes. Otherwise, both panes display the packages only.
	Show Modules	When the button is pressed, items in the tree view display under the corresponding module nodes. Otherwise, the project items display under their packages. Tip Grouping by modules enables you to show a package split among several modules.
	Show Module Groups	When the button is pressed, the items in the tree are arranged by user defined module groups .
	Group By Scope Type	When the button is pressed, the items in the tree are grouped by the type of scope, i.e. production, test or libraries.
	Show Illegals Only	When the button is pressed, the pane shows only illegal and invalid dependencies.
	Mark Illegal	Click this button to mark the selected dependency as illegal.
	Edit Rules	Click this button to open the Dependency Validation Dialog and define the rules for dependencies analysis.

Dependency Validation dialog

The Dependency Validation dialog box opens when you click the button on the Analyzed Code toolbar of the Dependency Viewer tool window.

Use the dialog box to declare rules that describe invalid dependencies.

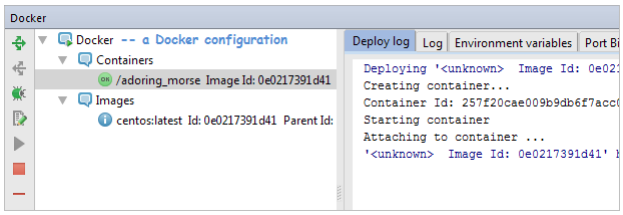
Item	Description
Deny usages of/ Allow usages of	Drop-down menus appear on pressing the Add button to the right. Use the drop-down menus to select the usage scope pattern: – Deny means that the specified scope code is recognized as invalid – Allow means that the specified scope code is recognized as valid. Scope pattern is defined in the Scopes dialog box, by clicking the ellipsis button.
in/ only in	Drop-down menus appear on pressing the Add button to the right. Use this drop-down menu to select the usage scope pattern where the previously selected usages are to be applied. For deny it means that the Deny usages of code are invalid if used within the in scope. For allow it means that the Allow usages of code is valid only in case if it is used within the only in scope. To set the scope pattern press the ellipsis button to call the Scopes dialog box.
Add	Click this button to add a line to the pattern list.
Remove	Click this button to remove the selected pattern line from the list.
Move up/ Move down	Click this button to move the selected pattern line up or down in the list.
Skip import statements	Select this check box to skip analyzing import statements. In this case references in the import statements are ignored.

Docker tool window

View | Tool Windows | Docker

For the tool window to be available, the Docker integration plugin must be installed and at least one Docker configuration must be defined.

The Docker tool window lets you manage your Docker images and containers.



All the available functions are accessed by means of the toolbar icons and context menu commands.

See also, Docker .

Icon	Command	Description
Docker configurations		
	Connect	Connect to Docker Remote API. As a result, the list of available Docker images and containers is shown.
	Disconnect	Disconnect from Docker Remote API. As a result, the list of Docker images and containers is hidden.
	Edit Configuration	Edit the Docker Remote API settings for the selected configuration.
	Deploy	Deploy an image or artifact using the deployment settings in one of the Docker Deployment run configurations. The run configuration is selected from the list that is shown.
	Debug	Start one of the Docker Deployment run configurations in the debug mode. The run configuration is selected from the list that is shown.
	Pull image	Pull an image from Docker Hub or other image repository, e.g. Quay . (The Pull Image dialog will open.)
Docker images		
	Create container	Create a container for the selected image according to an existing or new Docker Deployment run configuration.
	Delete image	Delete the selected image or images.
	Push image	Push the selected image to Docker Hub or other image repository, e.g. Quay . (The Push Image dialog will open.)
	Copy image ID	Copy an ID of the selected image to the clipboard.
Docker containers		
	(Re)deploy	Deploy an image or artifact using the deployment settings in the corresponding Docker Deployment run configuration.
	Debug	Start an associated run configuration in the debug mode.
	Edit Configuration	Edit the settings for an associated run configuration.
	Start container	Start the selected container.
	Stop container	Stop the selected container.
	Delete container	Delete the selected container or containers.
	Open browser	If a web app is deployed in the container: open the default web browser.
	Copy container ID	Copy an ID of the selected container to the clipboard.
	Copy image ID	Copy an ID of the associated image to the clipboard.
	Show log	Show the container log.
	Inspect	Show low-level container information in JSON format.
	Show processes	Show the list of processes running in the container.
	Attach	Open the console for working with standard streams (stdin, stdout and stderr).
	Exec	Run the `docker exec` command. For `docker exec [OPTIONS] CONTAINER COMMAND [ARG...]` , you should specify only the `COMMAND [ARG...]` part. See exec .

Documentation Tool Window

View | Tool Windows | Documentation

Ctrl+Q

This tool window appears in the following cases:

– Quick Documentation lookup is pinned by clicking in the upper-right side of the lookup window.

– Quick Definition tooltip is pinned by clicking in the upper-right corner of the tooltip.

As a result, the tool window appears in the list of available tool windows in the View menu, and gets the corresponding sidebar icon Documentation.

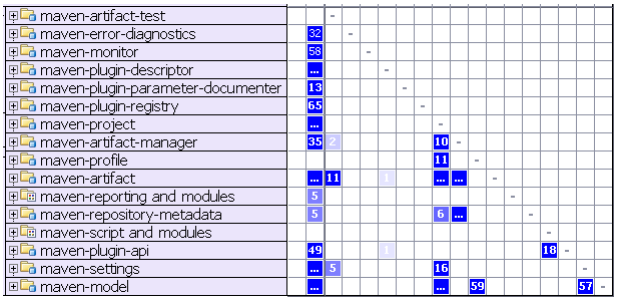
Icon	Shortcut	Description
	Left or Right	Switch to the previous or next documentation page (e.g. after using hyperlinks). Note On an OS X computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.
	Shift+F1	View external documentation in the default browser.
	F4	Switch to the item (e.g. source) that corresponds to the documentation page currently shown.
		Turn the Auto-update from source option on or off. When the option is on, the information in the tool window is synchronized with your navigation in the editor and other places in the UI.
		Click this icon to show the font size slider. Move the slider to increase or decrease the font size as required.

DSM Tool Window

View | Tool Windows | DSM

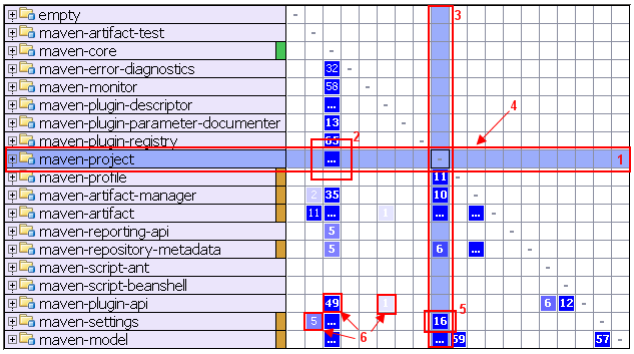
- This tool window becomes available after performing DSM analysis .
- Make sure that DSM Analysis plugin is installed and enabled. Refer to the Plugins page of the Settings dialog.

Here you can see the typical matrix view.



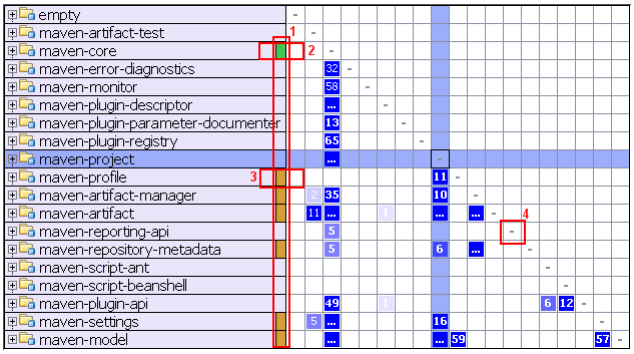
The row headers represent program structure. Everything is collapsed now and only modules are shown. When expanded, the header is tree-like, allowing you to expand modules and dig into program packages. * - node groups classes inside the package. The column headers are the same as the corresponding row headers. Thus they are not shown in order to save space. Instead, different visual aids are used on the row headers.

If you select a row, the matrix will look like this.



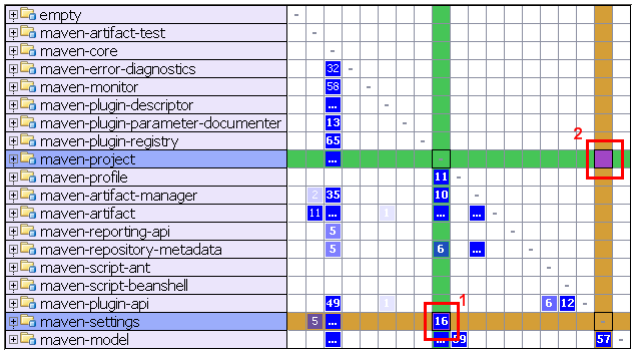
Here you can learn the following:

- The selected row and corresponding column are highlighted to visualize row dependencies.
- The ellipsis in the cell means that the `maven-core` module has many (more than 99) dependencies on `maven-project` module.
- The column shows the dependencies of the selected row.
- The row shows the dependencies on the selected row.
- This means that the `maven-project` module has 16 dependencies on `maven-settings` module.
- Various shades correspond to the number of dependencies.



- Color annotations help to visualize row dependencies at a glance.
- `maven-core` depends on `maven-project` .
- `maven-project` depends on `maven-profile` .
- The dashes on the diagonal correspond to self dependencies which are not shown.

You can select any cell to explore the dependencies indicated in it.

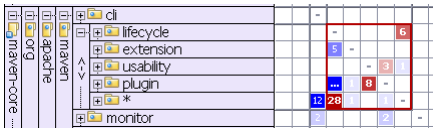


The cell #1 was selected. These color annotations mean that `maven-project` has 16 dependencies on `maven-settings`. The symmetrical cell (cell #2) shows dependencies in the other direction - in this case zero.

There is a simple mnemonic rule - all dependencies always *flow* from Green to Yellow.

Instead of alphabetically sorting rows, DSM view sorts dependencies in a special way: classes, which are used most are moved to the bottom. In a project with good structure this creates a triangular shape in the lower left half of the matrix.

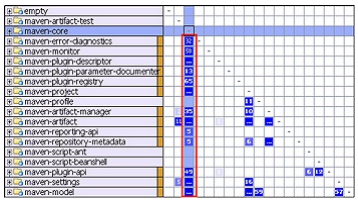
Cycles



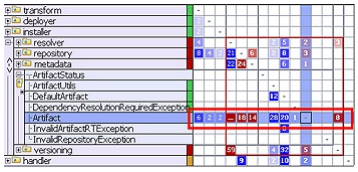
Mutual dependencies are shown in red. It means that the plugin and usability packages are both dependent on each other.

Patterns

There are two types of visual patterns. Vertical lines represent *aggregators*.



Horizontal lines appear in lowest-level or utility functionality.



Duplicates Tool Window

View | Tool Windows | Duplicates

The Duplicates tool window displays results of the search for duplicates.

On this page:

- Panes of the Duplicates tool window
- Left toolbar
- Upper toolbar
- Context menu commands

Panes of the Duplicates tool window

The window consists of the following panes:

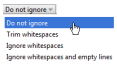
- The left pane displays the tree view of the duplicate fragments of source code. Each node shows the following information:
  - The number of duplicated code fragments found in scope.
  - The 'cost' of the duplicate (which is an arbitrary unit calculated using an additive algorithm on the base of the code block size; generally, the larger is the code fragment, the higher is its cost).
  - The containing class where the duplicates are located.
- The right pane shows the differences between the duplicated fragments of source code, selected in the left pane.

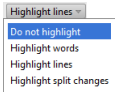
Left toolbar

Item	Shortcut	Description
	Rerun	Click this button to rerun the duplicates analysis in the active tab.
	Close Active Tab `Ctrl+Shift+F4`	Click this button to close the active tab.
	Autoscroll to Source	If the button is pressed, selecting an entry in the left pane opens the respective file in the editor.
	Eliminate Duplicates	Click this button to extract method from the duplicated code fragments. Refer to the Extract Method refactoring for details.
	`F1`	Click this button to show reference.

Upper toolbar

Item	Tooltip/Image/Shortcut	Description
	`F7` / `Shift+F7`	Move to the nxt/previous difference

Whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text.
		– Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default.
		– Trim whitespaces: `("\t", " ")` , if they appear in the end and in the beginning of a line.
		  – If two lines differ in trailing whitespaces only, these lines are considered equal.
		  – If two lines are different, such trailing whitespaces are not highlighted in the By word mode.
		– Ignore whitespaces: white spaces are not important, regardless of their location in the source code.
		– Ignore whitespaces and empty lines: the following entities are ignored:
		  – all whitespaces (as in the 'Ignore whitespaces' option)
		  – all added or removed lines consisting of whitespaces only
		  – all changes consisting of splitting or joining lines without changes to non-whitespace parts.
		
		For example, changing `a b c` to `a \n b c` is not highlighted in this mode.

Highlighting mode		Select the way differences granularity is highlighted. The available options are: – Highlight words: the modified words are highlighted – Highlight lines: the modified lines are highlighted – Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one. – Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	Jump to Source `F4`	Click this button to open the file in the active pane in the editor. The caret will be placed in the same position as in the Duplicates tool window.
	Synchronize scrolling	Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.
	Editor settings	Click this button to reveal the drop-down list of available check commands. Select or clear these check commands to show or hide line numbers, indentation guides, white spaces, and soft wraps. Note that same commands are duplicated in the context menu of the differences viewer gutter. Right-click the gutter to show the menu of check commands.

Context menu commands

Item	Keyboard Shortcut	Description

Jump to Source	F4	Open in the editor the file that contains the selected duplicate, and place the caret at the beginning of the duplicate. The fragment of code is highlighted.
Show Source	Ctrl+Enter	Open in the editor the file that contains the selected duplicate, and highlight the fragment of code.
Send to left/Send to right		Use these commands, or the arrow icons , to place the selected duplicate to the left or right pane of the differences viewer.

EJB Tool Window

View | Tool Windows | EJB

The EJB tool window presents the project structure as a tree where only modules with EJB facets are shown. Below each module node, the corresponding `ejb-jar.xml` deployment descriptor is shown.

Use this tool window to:

- Find out which modules have EJB facets.
- Create Enterprise Java Beans (EJB) .
- Edit the modules with EJB facets .
- Migrate to EJB 3.0 .

Context Menu

Item	Description
ER Diagram	For EJB nodes: use this command to open an ER diagram. (This command is unavailable if there are no entity beans in the selected module.)
Jump to Source (F4)	Select this option: - To open the selected file in the editor. (To open a file in the editor, you can, alternatively, double-click the file.) - To jump to the selected folder in the Project tool window.
New	Select this option to create an EJB . Select the necessary bean type in the submenu. The available options are: - Stateless Session Bean - Stateful Session Bean - Singleton Session Bean - Message Bean - BMP Entity Bean - CMP Entity Bean - EJB Relationship - Interceptor
Apply EJB 3.0 Style	Select this option to bring the structure of the beans in the selected module in compliance with the requirements of the EJB 3.0 specification .

Event Log

View | Tool Windows | Event Log

Event Log in the right-hand part of the bottom tool window bar

---

The Event Log tool window shows the information about "important" events that take place in IntelliJ IDEA.

The information about problematic situations (e.g. errors and exceptions) is displayed in red. In such cases, clicking the more link (if present) opens a balloon with a more detailed description of the error or exception. Clicking a description link (depending on the error, the text may be different e.g. NullPointerException) opens the IDE Fatal Errors dialog which lets you review the error and create a bug report.

- General tab
- Database tab
- Toolbar

General tab

The General tab appears when the Database tab opens. In the absence of the Database tab, the information is shown in the output pane which, in this case, is not tabbed.

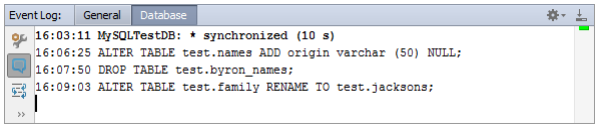
Database tab

Shown on the Database tab are the events related to working with the Database tool window and the Table Editor .

For the Database tool window, the following may be shown:

- The information about data synchronizations and various manipulations with databases such as creating, modifying and deleting tables and columns, etc.
- Error messages.

For the Table Editor only error messages are shown.



Toolbar

Item	Description
Settings	Use this icon to open the Notifications page where you can select which events you want to be notified of and also which events should be logged.
Show balloons	Use this icon to enable or disable showing notifications. Note that this icon is a toggle which turns the corresponding feature on or off. When showing notifications is enabled, you are notified of the events that take place in IntelliJ IDEA. The corresponding notifications are shown in the balloons. The alternative way to enable or disable this feature is by means of the Display balloon notifications check box on the Notifications page .
Use Soft Wraps	Use this icon to turn on or off the soft wrap mode for the output.
Scroll to the end	Use this icon to go to the end of the event log.
Mark all as read	Use this icon to mark all the messages as read.
Clear all	Use this icon to delete all the messages and thus clear the log. This function can alternatively be accessed by means of the context menu.
Help (F1)	Use this icon or shortcut to open the corresponding help topic.

Favorites Tool Window

View | Tool Windows | Favorites

`Alt+2`

The Favorites tool window lets you manage the following lists:

- Favorite project items . Each list of Favorites in the tool window appears with the star icon.
- Bookmarks . The list of bookmarks is marked with icon.
- Breakpoints . The list of breakpoints is marked with icon.

Initially, the lists are empty. So doing, the favorites list has the same name as the project.

The lists of bookmarks and breakpoints are filled in automatically, as the new bookmarks or breakpoints are added.

To add items to favorites, do one of the following:

- Select one or more items in the Project tool window or the Find tool window . Then choose File | Add To Favorites.
- Right-click an editor tab, and choose Add to Favorites or Add All to Favorites.

On this page:

- Toolbar buttons
- Context menu commands
- Title bar context menu and buttons
- Context menu of the side bar button
- Using drag-and-drop

Toolbar buttons

Item	Shortcut	Description
+	`Alt+Insert`	Use this button to create a new list of favorite items. In the Add New Favorites List dialog, specify the name for the new list and click OK.
	`Ctrl+Enter`	Depending on the selection, this button does one of the following: – If a Favorites list, or one of the nested favorites is selected, click this button to rename this list. – If a bookmark under the Bookmarks list is selected, click this button to change short description of the selected bookmark. – If a breakpoint under the Breakpoints list is selected, click this button to configure the selected breakpoint .
–	`Alt+Delete`	Use this button to delete the selected list or list item. Depending on the selection, the following behaviors take place: – If a Favorites list, or one of the nested favorites is selected, click this button to delete an entire favorites list, or the selected item. So doing an item or a list is deleted from the Favorites tool window, but is left intact in project. – If a bookmark under the Bookmarks list is selected, click this button to delete this bookmark from the list and from the source code. Vice versa, if a bookmark is deleted from the source code, it is also removed from the Favorites tool window. – If a breakpoint under the Breakpoints list is selected, click this button to delete the selected breakpoint from the list and from the source code. Vice versa, if a breakpoint is deleted from the source code, it is also removed from the Favorites tool window.

Context menu commands

When you right-click an item in the content pane, the context menu for this item is shown. This menu provides access to all the functions available for the selected item.

The following table lists and explains the commands for managing the favorites lists and their contents. Other commands, functionally, are similar to those in the Project tool window .

Item	Shortcut	Description
Rename Favorites List		Use this command to change the name of the selected list. In the New Name For Favorites List dialog, edit the name of the list and click OK.
Remove From Favorites	`N/A`	Use this command to delete the selected list or list item. Note that there may be list items corresponding to certain structures (for example, folders with subfolders and files inside). These may be deleted only as a whole, that is, you cannot delete their individual structural elements. To differentiate between the list items and their structural elements (if any), you can use the path information as a visual hint. Such information is shown for favorites list items but not for their elements.
Delete Favorites List '<list_name>'		Use this command to delete the list with the name `<list_name>` . As a rule, this is the list selected in the tool window.
Delete All User-Defined Lists Except '<list_name>'		Use this command to delete all user-defined favorites lists except the one with the name `<list_name>` . (The list with the name `<list_name>` , as a rule, is the one currently selected in the tool window.)
Add to Favorites		Use this command to add (i.e. copy) the selected list item to a different list. To add the item to an existing favorites list, select the target favorites list. To add the item to the list that does not yet exist, select Add To New Favorites List. In the Add New Favorites List dialog, specify the name of the new list and click OK. As a result, a new list is created and the selected item is added to this list.

Send To Favorites	Use this command to move the selected list item to a different list. To move the item to an existing favorites list, select the target favorites list. To move the item to the list that does not yet exist, select Send To New Favorites List. In the Add New Favorites List dialog, specify the name of the new list and click OK. As a result, a new list is created and the selected item is moved to this list.

Title bar context menu and buttons

The title bar context menu provides the options for controlling the tool window viewing modes . It also contains the commands for associating the tool window with a different tool window bar , resizing and hiding the tool window.

To access the menu, right-click the window title bar.

Note that most of the menu options may alternatively be accessed by means of the title bar buttons.

Toolbar icon	Context menu command	Description
	Pinned, Docked, Floating, Split Mode	These options let you control general appearance and behavior of the tool window, see Viewing Modes .
	Move to	To associate the tool window with a different tool window bar , select this command, and then select the destination tool window bar (Top, Left, Bottom or Right).
	Resize	To resize the tool window by moving one of its borders, select this command, and then select the necessary Stretch to option. Note that this command is not available in the floating mode.
 Shift+Escape	Hide	Use this command to hide the tool window. Tip When used in combination with the Alt key, clicking this button hides all the tool windows attached to the same tool window bar .
		Use this button to collapse all expanded nodes.

Context menu of the side bar button

Right-click the side bar button to reveal this context menu. Refer to the section Viewing Modes for the detailed information about the viewing modes.

Item	Description
	Use this command to turn the Show Members option on or off. If this option is on, the class members (fields, methods, etc.) are shown.
	Use this command to turn the Flatten Packages option on or off. If this option is off, the packages are shown as a hierarchy. If this option is on, all the packages appear at the same level and are identified by their qualified names.
	Use this command to turn the Compact Empty Middle Packages option on or off. If this option is off, the empty middle packages are shown as a hierarchy. If this option is on, all the packages appear at the same level and are identified by their qualified names.
	Use this command to turn the Autoscroll to Source option on or off. If this option is on, IntelliJ IDEA automatically navigates from a file (or a class member) selected in the Favorites tool window to the corresponding source file (or its fragment) in the editor. If the corresponding file is not currently open, it will open automatically.
	Use this command to turn the Autoscroll from Source option on or off. If this option is on, IntelliJ IDEA automatically navigates from a file (or a class member) selected in the editor, to the corresponding file in the Favorites tool window.

Using drag-and-drop

You can use drag-and-drop to:

- Add items to favorites: drag the item of interest from the Project tool window and drop it onto the desired favorites list in the Favorites tool window.
- Drag an external item from the Explorer/Finder and drop it onto the desired favorites list in the Favorites tool window.
- Move an item from one favorites list to another in the Favorites tool window.

Find Tool Window

View | Tool Windows | Find

Alt+3

On this page:

– Basics
– Toolbar buttons
– Context menu commands

Basics

Find tool window displays results of the following searches:

– Find/Replace in Path
– Find Usages
– Structural Search and Replace
– Refactoring Preview
– Find Usages of a data source , a table, or a column.
– The Usages pane appears in the Dependencies Viewer , when an entry is selected in the Parent Code pane.

The results of each search are displayed in a separate tab, or replace the contents of the current tab, depending on the Open in new tab dialog setting. By default the window appears at the bottom of the screen.

It has a toolbar with a set of buttons, a pane of results, and additional buttons for Replace in Path , Structural Replace, and Refactoring Preview operations.

Warning! Unless you find something, this tool window is not visible in the View menu.

Toolbar buttons

Item	Tooltip and shortcut	Description
	Options Ctrl+Shift+Alt+F7	Click this button to open one of the Find Usages dialogs, which corresponds to the symbol in question. You can edit the search settings and click Rerun button to execute the modified search query.
	Rerun Ctrl+F5	Rerun the last search. This button is not available for viewing code coverage results .
	Close Ctrl+Shift+F4	Close the current tab or the tool window. This button is not available in Replace in Path and Refactoring Preview dialogs.
	Pin	Use to pin or unpin the tab. If a tab is pinned, the results for the next command are shown on a new tab.
	Recent find usages Ctrl+E	Show the list of recent searches. Select an item in the list to see the search results.
 	Expand all Ctrl+NumPad Plus Collapse all Ctrl+NumPad –	Use these buttons to have all nodes expanded or collapsed.
	Previous/next occurrence Ctrl+Alt+Up Ctrl+Alt+Down	Navigate to the previous/next element in the tab of results.
	Autoscroll to source	Turns the Autoscroll to source option on or off. When the option is on and you select the search result, the corresponding source file opens in the editor and the appropriate fragment is highlighted in the file.
	Favorites	Click this button to add found usages to favorites .
	Export to Text File Alt+O	Save the contents of the current result tab. In the Export preview dialog, specify the target file or copy information to the clipboard. Before saving, you can also modify the information to be saved.
	Help F1	Use this icon or shortcut to open the corresponding help page.
	Group by usage type Ctrl+T	This button is available for Find Usages only. If this button is pressed, the search results are grouped by the following categories: – instanceof – import – cast target type – extends/implements clause

- class static member access
- method throws list
- .class
- field declaration
- local variable declaration
- method parameter declaration
- catch clause parameter declaration
- method return type
- delegates to another object instance
- delegates to another object instance with different parameters
- delegates to super
- delegates to super with different parameters
- recursive method calls
- string constants
- comments
- unclassified usage not related to any of the categories

	Group by test/production	If this button is pressed, the usages are grouped according to the Production and Test scopes.
	Group by module Ctrl+D	If this button is pressed, the found usages show under the corresponding module or library node. Tip This type of grouping is helpful, when a package is split between several modules.
	Group by package Ctrl+P	If this button is pressed, all the usages found are displayed under their respective packages.
	Group by file structure Ctrl+M	If this toggle is on, the found usages are shown under the corresponding method nodes.
	Merge usages from the same line Ctrl+F	If this toggle is on, the duplicate usages found on the same line are merged.
	Show read access Ctrl+R	This button is available for Find Usages only. If this button is pressed, the search results include references to the read access methods.
	Show write access Ctrl+W	This button is available for Find Usages only. If this button is pressed, the search results include references to the write access methods.
	Show import statements Ctrl+I	This button is available for Find Usages only. If this button is pressed, the search results include the usages in the import statements.
	Preview usages	Turns showing the Preview pane on or off.
	Sort members alphabetically	Click this button to have all members sorted alphabetically. Otherwise, members are sorted in the order they are declared.

Tip The following buttons are available for Replace in Path only.

Do Replace All	Alt+D	Click this button to replace all occurrences in the current tab of results.
Replace Selected	Alt+L	Click this button to replace the selected occurrence in the current tab of results.

Tip The following buttons are available for Refactoring Preview only.

Do Refactor	Alt+D	Click this button to perform refactoring on all occurrences in the current search results.
Cancel	Alt+C	Click this button to discard search results and close the results tab.

Tip The following buttons are available for Structural Replace only.

Replace All	Click this button to replace all found occurrences.
Replace Selected	Click this button to replace the highlighted occurrence.
Preview Replacement	Click this button to show how the changes will apply to the selected node in the Find tool window. In this case, the corresponding occurrence is highlighted in the source code.

Context menu commands

Item	Shortcut	Description
Jump to Source	F4	Navigate to the selected item in the source code.

Include	Insert	For an excluded item : include the item in the list of results.
Exclude	Delete	Exclude the selected item from the list of results. (Excluded items are shown strikethrough.) When you carry out the Replace All or the Do Refactor command, the excluded items are not affected.
Remove	Alt+Delete	Remove the selected item from the list of results.
Recent Find Usages	Ctrl+E	Show the list of recent searches. Select an item in the list to see the search results.
Add to Favorites		Add selected node to the Favorites list .

Framework Tool Window

View | Tool Windows | Framework

Warning! The tool window is available only in a Symfony2 or a Yii specific projects with the MVC view enabled .

In this tool window, examine the project in the terms of bundles, controllers, and other elements of the MVC pattern.

Grails Tool Window

View | Tool Windows | Grails

The special Grails tool window shows logical project structure displaying Controllers, Domain classes, Views, etc. as the Grails application elements rather than files and directories. From the Grails view, you can:

- Gain quick access to the application elements and their contents
- Navigate between the controller actions and associated views.
- Edit Groovy scripts from the `Scripts` directory

Item	Description
	Click this button to navigate from a file in the Editor that gets the focus, to the corresponding node (file, class, field, method, etc.) in the Project tool window.
	Click this button to collapse all nodes (`Ctrl+NumPad -`)
	Click this button to open the menu for configuring the current view and changing the tool window viewing modes. The menu items are options that you can turn on or off. An active item contains a check mark on the left side of its name. The available options in the menu are as follows: – Compact Empty Middle Packages or Hide Empty Middle Packages – Autoscroll to Source – Autoscroll from Source – Pinned, Docked, Floating, Windowed, Split Mode
	Click this button to hide the tool window(`Shift+Escape`). When used in combination with the Alt key, clicking this button hides all the tool windows attached to the same tool window bar.

Gradle Tool Window

View | Tool Windows | Gradle

The Gradle tool window displays Gradle tasks, and all changes made to the underlying `build.gradle` file, or to the IntelliJ IDEA project structure.

If there is no linked Gradle project, the window is disabled. You need to reimport your Gradle project to enable the Gradle tool window.

In this section:

– Gradle projects
– Gradle Tasks
– Toolbar
– Context Menu
– Context Menu Commands for Gradle Tasks

Gradle projects

Item	Description
	Click this button to open the menu for configuring the current view and changing the tool window viewing modes. The menu items are options that you can turn on or off. An active item contains a check mark on the left side of its name. The available options in the menu are as follows: – Group Tasks - select this option if you want to group your tasks. For example, build tasks will be included into the build directory. – Show Toolbar - select this option to show the toolbar for your Gradle projects. – Pinned, Docked, Floating, Windowed, Split Mode – Group Tabs - deselect this option to see the views on separate tabs if more than one view is available in a tool window. – Move to - select this option to move the Gradle tool window to either top, left or right. – Resize - select this option to resize the Gradle tool window.
	Click this button to hide the tool window(`Shift+Escape`). When used in combination with the Alt key, clicking this button hides all the tool windows attached to the same tool window bar.

Gradle Tasks

Item	Description
Run Configurations	This area contains a list of the last executed tasks and command types under which these tasks were executed.
Tasks	This area contains a list of all available tasks.
Dependencies	This area contains a list project dependencies.

Toolbar

Item	Description
	Click this button to refresh all registered Gradle projects after changes have been made to a Gradle script.
	Click this button to link a gradle project.
	Click this button to detach an external gradle project.
	Use this button to execute a Gradle task. When you click this icon, the Run Gradle Task dialog opens. You can enter the name of the task that you want to execute. Note that IntelliJ IDEA supports a code completion.
	Use these buttons to expand or collapse all the nodes.
	Click this button to import a module or a data to your gradle project via Project Data to Import dialog. It might be useful for the multi-module projects.
	Click this button to work with Gradle projects in the offline mode. It might be helpful when the network connection is slow or when you need to work offline.
	Click this button to configure the settings of the current Gradle project in the Gradle settings dialog box.

Context Menu

Item	Description
	Select this option or use a shortcut to open `build.gradle` file.
	Select this option to refresh external projects.
	Select this option to detach an external project.
Use auto-import	Select this option to enable the auto-import feature for your project.

Context Menu Commands for Gradle Tasks

Command	Description
Create <task>	Choose this command to create run/debug configuration for the selected Gradle task.
Run <task>	Choose this command to run the selected task with the task-specific run/debug configuration.
Debug <task>	Choose this command to debug the selected task with the task-specific run/debug configuration.
Execute before Make, Execute after Make	Choose these commands to set the respective flags for the selected task. In this case, *Before Make* and *After Make* comments appear next to the name of the node.

Execute before Rebuild, Execute after Rebuild	Choose these commands to set the respective flags for the selected task. In this case, *Before Rebuild* and *After Rebuild* comments appear next to the name of the node.
Execute before Run/Debug	Choose this command to specify a run/debug configuration, prior to which the selected task should be executed. In this case, *Before Run* comment appears next to the name of the node.
Assign Shortcut	Choose this command to associate the selected phase with a keyboard shortcut . In this case, the comment with the shortcut appears next to the name of the node.

Griffon Tool Window

View | Tool Windows | Griffon

The special Griffon tool window shows logical project structure displaying Controllers, Model classes, Views, etc. as the Griffon application elements rather than files and directories. From the Griffon view, you can gain quick access to the application elements and their contents.

Item	Description
	Click this button to navigate from a file in the Editor that gets the focus, to the corresponding node (file, class, field, method, etc.) in the Project tool window.
	Click this button to collapse all nodes (`Ctrl+NumPad -`)
	Click this button to open the menu for configuring the current view and changing the tool window viewing modes. The menu items are options that you can turn on or off. An active item contains a check mark on the left side of its name. The available options in the menu are as follows: – Compact Empty Middle Packages or Hide Empty Middle Packages – Autoscroll to Source – Autoscroll from Source – Pinned, Docked, Floating, Windowed, Split Mode
	Click this button to hide the tool window(`Shift+Escape`). When used in combination with the Alt key, clicking this button hides all the tool windows attached to the same tool window bar.

Grunt Tool Window

Context menu of a Gruntfile.js - Show Grunt Tasks

View | Tool Windows | Grunt

On this page:

- Accessing the Grunt Tool Window
- Building a Tree of Grunt Tasks
- Running Grunt Tasks and Targets
- Toolbar
- Context Menu of a Tree
- Context Menu of a Task or a Target

Accessing the Grunt Tool Window

View | Tool Windows | Grunt

- the tool window can be accessed this way only after you have opened it using the Show Grunt Tasks command.

The tool window is available only when:

1. The Node.js runtime environment is installed on your computer.
2. The NodeJS repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. The `grunt-cli` package is installed globally and the `grunt` package is installed in the current project, see Installing Grunt for details.
4. At least one `Gruntfile.js` file is available in the current project.

The tool window opens when you invoke Grunt by choosing Show Grunt Tasks on the context menu of a `Gruntfile.js` in the Project tool window or of a `Gruntfile.js` opened in the editor.

As soon as you invoke Grunt, it starts building a tree of tasks according to the `Gruntfile.js` on which it was invoked. If a task has targets , the task is displayed as a node and the targets are listed under it.

If you have several `Gruntfile.js` files in your project, you can build a separate tasks tree for each of them and run tasks without abandoning the previously built tasks trees. Each tree is shown in a separate tab.

Building a Tree of Grunt Tasks

To build a tasks tree, do one of the following:

- Select the required `Gruntfile.js` file in the Project tool window and choose Show Grunt Tasks on the context menu of the selection.
- Open the required `Gruntfile.js` file in the editor and choose Show Grunt Tasks on the context menu of the editor.
- If the Grunt tool window is already opened, click on the toolbar and choose the required `Gruntfile.js` file from the list. IntelliJ IDEA adds a new node and builds a tasks tree under it. The title of the node shows the path to the `Gruntfile.js` file according to which the tree is built.

Running Grunt Tasks and Targets

To run a task or a target from the tree, do one of the following:

- Double click the required task or target in the tree.
- Select the required task or target and choose Run <task name> on the context menu of the selection.
- Select the required task or target and press `Enter`.
- To run the `default` task, select the root node in the tree and choose Run default on the context menu of the selection.
- To navigate to the definition of a task or target, select the required task or target in the tree and choose Jump to source on the context menu of the selection.

The task or target execution output will be displayed in the Run tool window . The name of the target is shown in the format `<task name>:<target name>` . The tool window shows the Grunt output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

To run several tasks or targets, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required tasks or targets, then choose Run on the context menu of the selection.

Toolbar

Item	Tooltip	Description
	Add Gruntfile	Click this button to have a tasks tree for another `Gruntfile.js` file built. Choose the required `Gruntfile.js` file from the pop-up list. IntelliJ IDEA adds a new node and builds a tasks tree under it.
	Remove Gruntfile	Click this button to remove the tasks tree under the selected node.
	Reload tasks	Click this button to have the tasks tree under the selected node re-built. You may need a tree re-built after updating the corresponding `Gruntfile.js` file because Grunt does not apply changes to trees on the fly.

Context Menu of a Tree

Item	Description
Grunt Settings	Choose this menu item to open the Grunt Settings dialog box and view or edit the Node.js configuration
Jump to Source	Choose this menu item to open the `Gruntfile.js` file for which the current tree is built.

Reload tasks	Choose this menu item to have the tree of tasks under the selected node re-built.
Copy Path	Choose this menu item to save the path to the `Gruntfile.js` file according to which the current tree was built to the clipboard.
Remove Gruntfile.js	Choose this menu item to remove the tree of tasks under the selected node.

Context Menu of a Task or a Target

Item	Description
Run <task/target name>	Choose this menu item to run the selected task or target.
Debug <task/target name>	Choose this menu item to debug the selected task or target.
Edit <task/target name> settings	Choose this menu item to open the Run/Debug Configuration dialog box and edit the predefined settings for the selected task or target.
Jump to Source	Choose this menu item to open the `Gruntfile.js` file for which the current tree is built and navigate to the definition of the selected task or target.

Gulp Tool Window

On this page:

- Accessing the Gulp Tool Window
- Building a Tree of Gulp Tasks
- Running Gulp Tasks
- Toolbar
- Context Menu of a Tree
- Context Menu of a Task or a Target

Accessing the Gulp Tool Window

Context menu of a Gulpfile.js - Show Gulp Tasks

View | Tool Windows | Gulp - the tool window can be accessed this way only after you have opened it using the Show Gulp Tasks command or after you have run tasks through the Gulp.js run configuration.

The tool window is available only when:

1. The Node.js runtime environment is installed on your computer.
2. The NodeJS repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. The `gulp` package is installed in the current project, see Installing Gulp for details.
4. At least one `Gulpfile.js` file is available in the current project.

The tool window opens when you invoke Gulp.js by choosing Show Gulp Tasks on the context menu of a `Gulpfile.js` in the Project tool window or of a `Gulpfile.js` opened in the editor. The tree is built according to the `Gulpfile.js` file on which Gulp.js was invoked. If you have several `Gulpfile.js` files in your project, you can build a separate tasks tree for each of them and run tasks without abandoning the previously built tasks trees. Each tree is shown under a separate node.

If you have several `Gulpfile.js` files in your project, you can build a separate tasks tree for each of them and run tasks without abandoning the previously built tasks trees. Each tree is shown under a separate node.

Building a Tree of Gulp Tasks

To build a tree of tasks, do one of the following:

- Select the required `Gulpfile.js` file in the Project tool window and choose Show Gulp tasks on the context menu of the selection.
- Open the required `Gulpfile.js` file in the editor and choose Show Gulp tasks on the context menu of the editor.
- If the Gulp tool window is already opened, click on the toolbar and choose the required `Gulpfile.js` file from the list. IntelliJ IDEA adds a new node and builds a tasks tree under it. The title of the node shows the path to the `Gulpfile.js` file according to which the tree is built.

Running Gulp Tasks

To run a task, do one of the following:

- Double click the required task in the tree.
- Select the required task and choose Run <task name> on the context menu of the selection.
- Select the required task and press `Enter`.
- To run the `default` task, select the root node in the tree and choose Run default on the context menu of the selection.
- To navigate to the definition of a task, select the required task in the tree and choose Jump to source on the context menu of the selection.

The task execution output will be displayed in the Run tool window .

To run several tasks, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required tasks, then choose Run on the context menu of the selection.

Toolbar

Item	Tooltip	Description
	Add Gulpfile	Click this button to have a tasks tree for another `Gulpfile.js` file built. Choose the required `Gulpfile.js` file from the pop-up list. IntelliJ IDEA builds a tasks tree and shows it under a separate node.
	Remove Gulpfile	Click this button to remove the tasks tree under the selected node.
	Reload tasks	Click this button to have the tasks tree under the selected node re-built. You may need a tree re-built after updating the corresponding `Gulpfile.js` file because Gulp.js does not apply changes to trees on the fly.

Context Menu of a Tree

Item	Description
Gulp Settings	Choose this menu item to open the Gulp Settings dialog box and view or edit the Node.js configuration
Jump to Source	Choose this menu item to open the `Gulpfile.js` file for which the current tree is built.
Reload tasks	Choose this menu item to have the tree of tasks under the selected node re-built.
Copy Path	Choose this menu item to save the path to the `Gulpfile.js` file according to which the current tree was built to the clipboard.
Remove Gulpfile.js	Choose this menu item to remove the tree of tasks under the selected node.

Context Menu of a Task or a Target

Item	Description

Run <task name>	Choose this menu item to run the selected task.
Debug <task name>	Choose this menu item to debug the selected task.
Edit <task name> settings	Choose this menu item to open the Run/Debug Configuration dialog box and edit the predefined settings for the selected task.
Jump to Source	Choose this menu item to open the `Gulpfile.js` file for which the current tree is built and navigate to the definition of the selected task.

Hibernate Console Tool Window

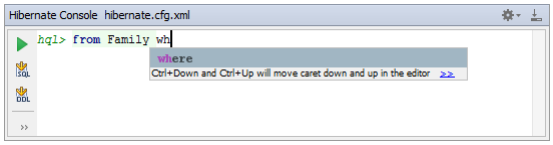
From the Persistence tool window (for a session factory or any node within it):

– on the title bar

– Console from the context menu

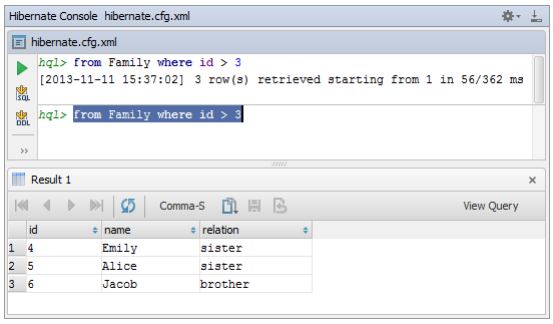
– `Ctrl+Shift+F10`

When you open the Hibernate Console tool window, first, the input pane opens. This is where you compose your HQL queries.

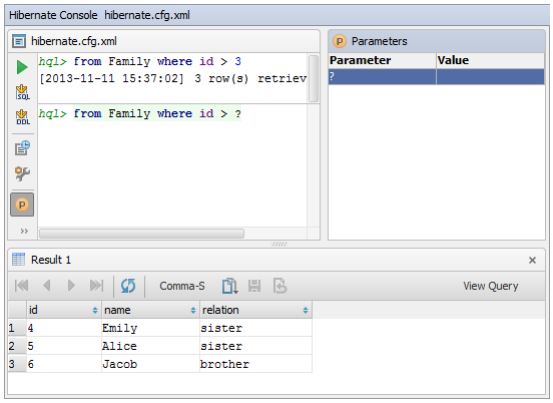


When you run your first query ( ), the output pane opens above the input pane. Basically, this is the log of operations performed in the console.

If your query retrieves data (e.g. `from` , `select`), also the Result pane opens showing the retrieved data in table format.



Additionally, you can open the Parameters pane ( ) to manage parameters in your queries.



On this page:

– Toolbar icons and shortcuts

– Output pane

– Result pane

– Parameters pane

See also, Working with the Hibernate Console .

Toolbar icons and shortcuts

Most of the functions in the Hibernate Console tool window are accessed by means of the toolbar icons and associated keyboard shortcuts.

Item	Shortcut	Description
Execute Query	`Ctrl+Enter`	Use this icon or shortcut to run the current query.
Generate SQL	`Ctrl+Shift+Enter`	Use this icon or shortcut to generate an SQL equivalent of the current query. The generated SQL statement will be shown in the output pane.
Generate DDL	`Ctrl+Shift+Alt+Enter`	Use this icon or shortcut to generate DDL SQL statements (`CREATE TABLE` , `ALTER TABLE` and `DROP TABLE`) for all the objects (classes) associated with the corresponding session factory. The generated statements will be shown in the output pane.
	`Ctrl+Alt+E`	Use this icon or shortcut to open a dialog that shows all the queries that you have run in the console. See also, Running auto-memorized queries.
Settings		Use this icon to open the Database page of the Settings dialog to view or edit the settings for the database, Hibernate and JPA consoles, the Table Editor and the Database tool window.
View Parameters		Use this icon to open or close the Parameters pane .
Restore Layout		Use this icon to restore the original tool window layout (after the rearrangements that you have made).

Terminate Process	Ctrl+F2	Use this icon or shortcut to terminate execution of the current query.
Close	Ctrl+Shift+F4	Use this icon or shortcut to close the console.

Output pane

This pane shows the queries that you have run as well as the information about their execution. When errors occur, the corresponding information is also shown in this pane.

For most of the events the following information is provided:

- Timestamp, that is, when the event took place.
- For data manipulation operations - how many rows were affected (e.g. changed or deleted). For data retrieval operations - how many rows were retrieved.
- Duration in milliseconds.

Use the following context menu commands:

- Copy (Ctrl+C) to copy the text selected in the output pane to the clipboard.
- Compare with Clipboard to compare the text selected in the output pane with the contents of the clipboard.
- Clear All to clear all the contents of the output pane.

Result pane

This pane shows the data retrieved from the database in table format. You can sort the data as well as perform other, associated tasks.

- Main functions
- Using the header row

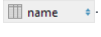
Main functions. Most of the functions in the Result pane are accessed by means of controls on the toolbar, context menu commands for the data cells, and associated keyboard shortcuts.

Item	Shortcut	Description
and		These icons and corresponding commands are for switching between the result set pages, i.e. the pages that show the retrieved data. A fixed number of rows shown simultaneously is referred to as a result set page. If this number is less than the number of rows that satisfy the query, only a subset of all the rows is shown at a time. In such cases, you can use and to switch between the subsets. (If all the rows are currently shown, these icons and the corresponding commands are inactive.) The result set page size is set on the Database page of the Settings dialog.
First Page		Use this icon or command to switch to the first of the result set pages to see the first series of rows.
Previous Page	Ctrl+Alt+Up	Use this icon, command or shortcut to switch to the previous result set page to see the previous series of rows.
Next Page	Ctrl+Alt+Down	Use this icon, command or shortcut to switch to the next result set page to see the next series of rows.
Last Page		Use this icon or command to switch to the last of the result set pages to see the last series of rows.
Reload Page	Ctrl+F5	Use this icon, command or shortcut to refresh the current table view. Use this function to: - Synchronize the data shown with the actual contents of the database. - Apply the Result set page size setting after its change.
Tab S...(TSV) Data Extractor		Use this button or command to open a menu in which you can select an output format for your data. The Configure CSV Formats command in that menu opens the CSV Formats Dialog that lets you manage the delimiter-separated values formats (e.g. CSV, TSV).
Copy All To Clipboard / Save All To File		- Copy All To Clipboard. Use this command to copy the table data onto the clipboard. - Save All To File. Use this command to save the table data in a file. In the dialog that opens, specify the location and name of the file. Note: - The data extractor which is currently active is applied. - If only a subset of corresponding rows is currently shown, all the appropriate rows are copied to the clipboard or saved in a file anyway. (The number of rows currently shown may be limited by the Result set page size parameter.)
Save LOB		Use this command to save the large object (LOB) currently selected in the table in a file.
Reset View		Use this command to restore the initial table view after reordering or hiding the columns, or sorting the data. As a result, the data, generally, becomes unsorted, the columns appear in the order they are defined in the corresponding query, and all the columns are shown.
View Query		Use this button to view the query which was used to generate the table. To close the pane where the query is shown, press Escape .
Go To | Row	Ctrl+G	Use this command or shortcut to switch to a specified row. In the dialog that opens, specify the row number to go to.

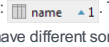
Using the header row. In the Result pane, you can use the cells in the header row (i.e. the row where column names are shown) for:

- Sorting data
- Reordering columns
- Hiding and showing columns

You can sort table data by any of the columns by clicking the cells in the header row.

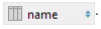
Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: 

. The sorting marker in this case indicates that the data is not sorted by this column.

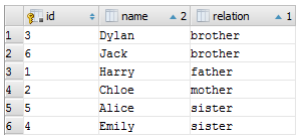
If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance: 

. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: .

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled:  .

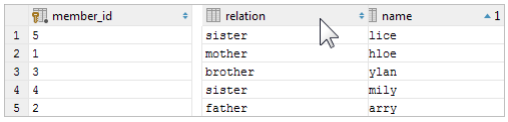
Here is an example of a table where data are sorted by two of its columns.



	id	name ▲2	relation ▲1
1	3	Dylan	brother
2	6	Jack	brother
3	1	Harry	father
4	2	Chloe	mother
5	5	Alice	sister
6	4	Emily	sister

To restore the initial "unsorted" state for the table, click and select Reset View.

To reorder columns, use drag-and-drop for the corresponding cells in the header row. To restore the initial order of columns, click and select Reset View.



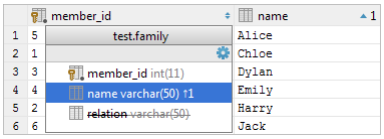
	member_id	relation	name ▲1
1	5	sister	lice
2	1	mother	hloe
3	3	brother	ylan
4	4	sister	mily
5	2	father	arry

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
   - Right-click any of the cells in the header row and select Column List.
   - Press `Ctrl+F12`.

   In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

To show all the columns, click and select Reset View.

See also, Using the Structure view to sort data, and hide and show columns.

Parameters pane

The Parameters pane shows the parameters detected in the input pane and lets you edit their values. To open or close this pane, use on the toolbar.

Parameter values can be specified just as text or numbers, or as Groovy expressions that contain object references and method calls. For example, the value for the `date` parameter in the query

```
SELECT o
                    FROM Order o
                    WHERE o.date >
                    :date
```

could be specified as

```
new java.sql.Date(System.currentTimeMillis() - 24*3600*1000)
```

To start editing a value, switch to the corresponding table cell and start typing. To indicate that you have finished editing a value, press `Enter` or switch to a different cell. To quit the editing mode and restore an initial value, press `Escape`.

When you select a row in the table, the corresponding parameter is highlighted in the input pane.

Hierarchy Tool Window

View | Tool Windows | Hierarchy

Alt+8

Use this tool window to analyze and navigate through hierarchies of classes , calls, methods .

Warning! The tool window is available only after you have built a hierarchy for the first time.

The contents of the tool window are not automatically updated as you navigate through the source code or switch between the editor tabs. The tool window shows the results of the latest hierarchy command and is updated when you run the next hierarchy command, unless the tab with one of the previously built hierarchies is pinned .

Toolbar buttons

Item	Description	Available In
	When this button is pressed, the hierarchical tree shows both the parent and child classes of the selected class, which is marked with an arrow in the tree of results. Tip For the interfaces this button is disabled.	Class hierarchies
	Depending on the hierarchy type: – For class hierarchies - when this button is pressed, the tool window shows all classes that extend the selected class. – For call hierarchies - when this button is pressed, the tool window shows the callees of the selected method.	Class hierarchies Call hierarchies
	Depending on the hierarchy type: – For class hierarchies - when this button is pressed, the tool window shows the hierarchy of each supertype of the current class. – For call hierarchies - when this button is pressed, the tool window shows the callers of the selected method.	Class hierarchies Call hierarchies
	When this button is pressed, the elements within a tree are sorted alphabetically.	All hierarchies
Scope	Use this drop-down list to limit the scope of the current hierarchy: – Project - IntelliJ IDEA traces usages of the method across the project. – Test - IntelliJ IDEA traces usages of the method across the test classes. – All - IntelliJ IDEA traces usages of the method across the project and the libraries. – This class - the scope is limited to the current class. In addition to the pre-configured scopes, you can define your own one: select Configure from the drop-down list, and define the required scope in the Scopes dialog box that opens.	Call hierarchies
In a method hierarchy, the tree-views of the following classes are available: – + - where this method is defined. – — - where this method is not defined, but defined in the superclass. – ! - where this method should be defined, because the class is not abstract.		
	If you made any changes of the classes or the class structure, they become visible in the Hierarchy tool window only after you press this button.	All hierarchies
	Toggle the Autoscroll to source mode. When the button is pressed, every time the node is focused, the corresponding line of source code is highlighted in the editor.	All hierarchies
	When this button is pressed, the current tab will not be overwritten; instead, the results of the next command will be displayed in a new tab.	All hierarchies
	Click this button to export a hierarchy into a text file in the specified location.	All hierarchies
	Click this button to close the selected tab of results.	All hierarchies
	Click this button to show reference page.	All hierarchies

Inspection Tool Window

View | Tool Windows | Inspection

The Inspection tool window displays inspection results on separate tabs.

The left-hand pane of each tab shows a tree view of the inspections for which problems are found. The right-hand pane shows summary information for an item selected in the left-hand pane.

- You can access the tool window this way only when it is already opened through Analyse | Inspect Code .
- After you deactivate the tool window manually by clicking the Close button, the tool window is again available only through Analyze | Inspect Code.

On this page:

- Toolbar buttons
- Context menu commands
- Inspection report

Toolbar buttons

Item	Shortcut	Description
	Ctrl+F5	Click this button to run the inspection and show the results on the same tab.
	Ctrl+Shift+F4	Click this button to close the current tab or the tool window.
	Ctrl+NumPad Plus	Expand all nodes
	Ctrl+NumPad -	Collapse all nodes
	Ctrl+Alt+Down	Navigate to the next item.
	Ctrl+Alt+Up	Navigate to the previous item.
	F1	Use this icon or shortcut to open the corresponding help page.
		If this toggle is on, the problems are grouped into Errors and Warnings. Otherwise, the problems are grouped by inspections.
		Turns grouping by directories on or off.
		When this toggle is on, the resolved and excluded items are not shown.
		Turns the Autoscroll to source option on or off. When the option is on and you select an item, the corresponding source file opens in the editor and the appropriate fragment is highlighted.
		Click this button to export the inspection results into XML or HTML format .
		Click this button to edit the current inspection settings .
	Alt+Enter	Click this button to resolve the problem for the selected inspection item by choosing one of the available quick fixes from the list.

Context menu commands

Item	Shortcut	Description
Jump to Source	F4	Open the file that contains the selected problem in the editor and place the cursor at the beginning of the corresponding code fragment.
Exclude	Delete	Exclude the selected items from further examination. Excluded nodes are shown strikethrough. If the filter toggle is on, the excluded nodes are hidden.
Include	Insert	Include previously excluded items in the list of results. All nested subelements are included too.
	Alt+Enter	Select one of the suggested solutions.
Suppress problem / Suppress problem for class		Suppress the inspection for the selected problem or the selected class.
Edit Settings		Change the settings for the selected inspection or group of inspections in the Errors dialog.
Disable inspection		Disable alerts for the selected inspection in the active tab of results. If the filter toggle is on, the nodes for disabled inspections are hidden. See also, Disabling and Enabling Inspections .
Run inspection on		Rerun the selected inspection and display the results on a new tab.

Inspection report

The inspection report is shown in the right-hand pane of the results tab when an inspection node is selected in the left-hand pane. The report may include the following:

- Problem resolution: A button for each of the available solutions. Clicking a button invokes the corresponding fix. If no buttons are present, you have to fix the problem yourself.
- Suppress: Click this button to reveal the list of inspection suppress options.
- Problem synopsis: A brief description of the problem.
- Disable inspection: Disable alerts for the selected inspection in the active tab of results. If the filter toggle is on, the nodes for disabled inspections are hidden.
  See also, Disabling and Enabling Inspections .
- Run inspection on...: Rerun the selected inspection and display the results on a new tab.

Java EE: App Tool Window

View | Tool Windows | JavaEE:App

This window is available only for modules with a JavaEE Application facet .

The JavaEE: App tool window provides a tree view of your JavaEE Application facet resources. Below each module node, the corresponding `application.xml` deployment descriptor is shown.

Context Menu

Item	Description
Jump to Source (F4)	Select this option: – To open the selected file in the editor. (To open a file in the editor, you can, alternatively, double-click the file.) – To jump to the selected folder in the Project tool window.

Java Enterprise Tool Window

To open this tool window:

View | Tool Windows | Java Enterprise

See also, Showing a tool window.

The Java Enterprise tool window lets you look at your project from the Java EE perspective.

The tool window includes a set of panes where the "Java EE contents" of a selected item are shown in a pane to the right. If appropriate, the Javadoc HTML documentation (or, sometimes, a diagram) is shown for an item in the rightmost pane.

The toolbar buttons are used to show or hide categories of items (e.g. the modules or CDI beans), or to change the way the items are shown.

The context menus include commands for switching between the panes (Previous or Next) and opening items in the editor (Edit).

You can filter the information in most of the panes by specifying the corresponding search string.

– Toolbar icons
– Context menu commands
– Filtering information

Toolbar icons

With very few exceptions, the toolbar icons are toggles that can be on or off.

The first two icons (and) are available always. The rest of the icons appear only for certain Java EE technologies and frameworks (e.g. CDI, WebSocket).

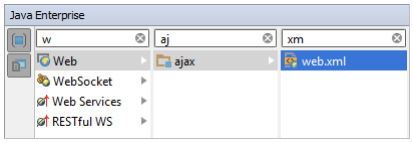
Icon	Description
	The state of the icon defines the level on which the Java EE technologies and frameworks are shown. When on, the technologies are shown on the first level (i.e. in the leftmost pane). In this case, if is on, and the modules are shown in the pane to the right, you can select a technology and see, for example, in which modules the selected technology is used. When off, the technologies are shown on the second level. That is, your modules appear in the leftmost pane, and the technologies in the pane to the right. In this case, you can select a module and see which of the technologies are used in the selected module.
	If is on, this icon is used to show or hide the modules. In this case, the modules, if shown, appear in the second pane from the left.
	For CDI: Use this icon to show or hide CDI beans.
	For CDI: Use this icon to show or hide producer methods and fields.
	Depending on the technology: – For CDI: Use this icon to switch between showing a diagram or documentation in the rightmost pane. If on, the dependency injection diagrams are shown for selected items. Otherwise, the Javadoc HTML documentation is shown. – For WebSocket: Use this icon to change the way the endpoint classes are shown and ordered. If off, the information for the classes is shown in the following order: the class name first and then the endpoint URL (if available). The classes in this case are ordered by their names. If on, the endpoint URLs are shown first and are followed by the class names. The classes in this case are ordered by the endpoint URLs. – For RESTfull Web Services: use this icon to change the way resource classes and their methods are shown and ordered. If off, the information for the classes and methods is shown in the following order: the names first and then the resource URLs. The classes and methods in this case are ordered by their names. If on, the resource URLs are shown first and the information is ordered by the URLs. Note that the resource classes may be shown or hidden by means of . To show or hide the methods, use .
	For CDI: Use this icon to show or hide the `InjectionPoint` types. Note that if is on, the injection points are shown on the diagrams in the rightmost pane, and this icon becomes inactive.
	For WebSocket: Use this icon to show or hide the classes annotated with `@ServerEndpoint` .
	For WebSocket: Use this icon to show or hide the classes annotated with `@ClientEndpoint` .
	For RESTful Web Services: Use this icon to show or hide the resource classes. If off, there is a pane showing the classes. When you select a class, its methods are shown in the pane to the right. So, in this case, the methods are grouped by the classes in which they are defined. If on, the classes are not shown. The methods belonging to different classes are all shown in the same pane and at the same time.
	For RESTful Web Services: Click this icon and then the necessary option to show or hide the methods annotated with `@GET` , `@POST` , `@PUT` , or `@DELETE` .

Context menu commands

Command	Shortcut	Description
Previous	`Left`	Use this command or shortcut to switch to the pane to the left.
Next	`Right`	Use this command or shortcut to switch to the pane to the right.
Edit	`Enter`	Use this command or shortcut to switch to the editor to view or edit the source code for the selected item. For a module, this command results in showing the module settings in the Project Structure dialog .

Filtering information

When in the pane of interest, just start typing. As a result, only the items whose names contain the specified string will be shown.



JPA Console Tool Window

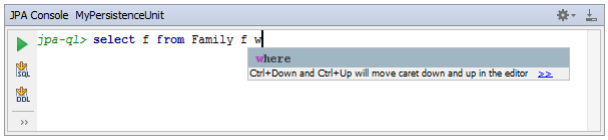
From the Persistence tool window (for a persistence unit or any node within it):

– on the title bar

– Console from the context menu

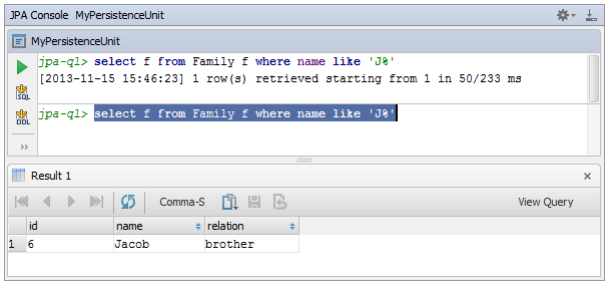
– `Ctrl+Shift+F10`

When you open the JPA Console tool window, first, the input pane opens. This is where you compose your JPQL queries.

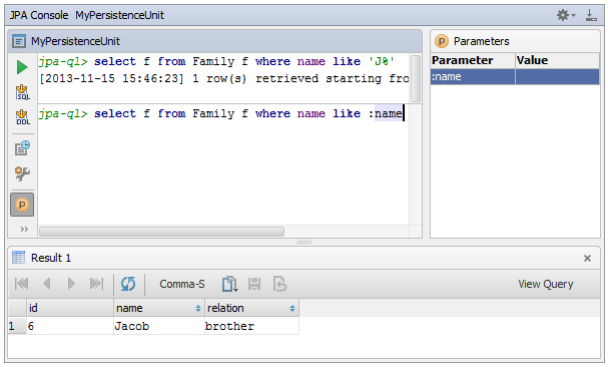


When you run your first query (▶), the output pane opens above the input pane. Basically, this is the log of operations performed in the console.

If your query retrieves data (e.g. `select`), also the Result pane opens showing the retrieved data in table format.



Additionally, you can open the Parameters pane (Ⓟ) to manage parameters in your queries.



On this page:

– Toolbar icons and shortcuts

– Output pane

– Result pane

– Parameters pane

See also, Working with the JPA Console .

Toolbar icons and shortcuts

Most of the functions in the Hibernate Console tool window are accessed by means of the toolbar icons and associated keyboard shortcuts.

Item	Shortcut	Description
▶ Execute Query	`Ctrl+Enter`	Use this icon or shortcut to run the current query.
Generate SQL	`Ctrl+Shift+Enter`	Use this icon or shortcut to generate an SQL equivalent of the current query. The generated SQL statement will be shown in the output pane.
Generate DDL	`Ctrl+Shift+Alt+Enter`	Use this icon or shortcut to generate DDL SQL statements (`CREATE TABLE` , `ALTER TABLE` and `DROP TABLE`) for all the objects (classes) associated with the corresponding persistence unit. The generated statements will be shown in the output pane.
	`Ctrl+Alt+E`	Use this icon or shortcut to open a dialog that shows all the queries that you have run in the console. See also, Running auto-memorized queries.
Settings		Use this icon to open the Database page of the Settings dialog to view or edit the settings for the database, Hibernate and JPA consoles, the Table Editor and the Database tool window.
Ⓟ View Parameters		Use this icon to open or close the Parameters pane .
Restore Layout		Use this icon to restore the original tool window layout (after the rearrangements that you have made).
Terminate Process		Use this icon or shortcut to terminate execution of the current query.
Close	`Ctrl+Shift+F4`	Use this icon or shortcut to close the console.

Output pane

This pane shows the queries that you have run as well as the information about their execution. When errors occur, the corresponding information is also shown in this pane.

For most of the events the following information is provided:

- Timestamp, that is, when the event took place.
- For data manipulation operations - how many rows were affected (e.g. changed or deleted). For data retrieval operations - how many rows were retrieved.
- Duration in milliseconds.

Use the following context menu commands:

- Copy (`Ctrl+C`) to copy the text selected in the output pane to the clipboard.
- Compare with Clipboard to compare the text selected in the output pane with the contents of the clipboard.
- Clear All to clear all the contents of the output pane.

Result pane

This pane shows the data retrieved from the database in table format. You can sort the data as well as perform other, associated tasks.

- Main functions
- Using the header row

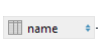
Main functions. Most of the functions in the Result pane are accessed by means of controls on the toolbar, context menu commands for the data cells, and associated keyboard shortcuts.

Item	Shortcut	Description
and		These icons and corresponding commands are for switching between the result set pages, i.e. the pages that show the retrieved data. A fixed number of rows shown simultaneously is referred to as a result set page. If this number is less than the number of rows that satisfy the query, only a subset of all the rows is shown at a time. In such cases, you can use , and to switch between the subsets. (If all the rows are currently shown, these icons and the corresponding commands are inactive.) The result set page size is set on the Database page of the Settings dialog.
First Page		Use this icon or command to switch to the first of the result set pages to see the first series of rows.
Previous Page	`Ctrl+Alt+Up`	Use this icon, command or shortcut to switch to the previous result set page to see the previous series of rows.
Next Page	`Ctrl+Alt+Down`	Use this icon, command or shortcut to switch to the next result set page to see the next series of rows.
Last Page		Use this icon or command to switch to the last of the result set pages to see the last series of rows.
Reload Page	`Ctrl+F5`	Use this icon, command or shortcut to refresh the current table view. Use this function to: - Synchronize the data shown with the actual contents of the database. - Apply the Result set page size setting after its change.
Tab S...(TSV) Data Extractor		Use this button or command to open a menu in which you can select an output format for your data. The Configure CSV Formats command in that menu opens the CSV Formats Dialog that lets you manage the delimiter-separated values formats (e.g. CSV, TSV).
Copy All To Clipboard / Save All To File		- Copy All To Clipboard. Use this command to copy the table data onto the clipboard. - Save All To File. Use this command to save the table data in a file. In the dialog that opens, specify the location and name of the file. Note: - The data extractor which is currently active is applied. - If only a subset of corresponding rows is currently shown, all the appropriate rows are copied to the clipboard or saved in a file anyway. (The number of rows currently shown may be limited by the Result set page size parameter.)
Save LOB		Use this command to save the large object (LOB) currently selected in the table in a file.
Reset View		Use this command to restore the initial table view after reordering or hiding the columns, or sorting the data. As a result, the data, generally, becomes unsorted, the columns appear in the order they are defined in the corresponding query, and all the columns are shown.
View Query		Use this button to view the query which was used to generate the table. To close the pane where the query is shown, press `Escape` .
Go To | Row	`Ctrl+G`	Use this command or shortcut to switch to a specified row. In the dialog that opens, specify the row number to go to.

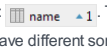
Using the header row. In the Result pane, you can use the cells in the header row (i.e. the row where column names are shown) for:

- Sorting data
- Reordering columns
- Hiding and showing columns

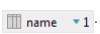
You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: 

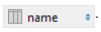
. The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance: 

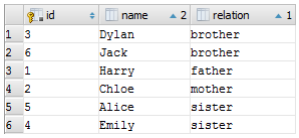
. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: 

.

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled:  .

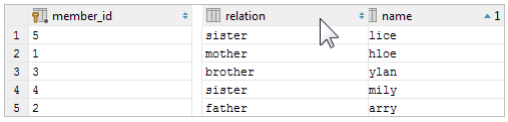
Here is an example of a table where data are sorted by two of its columns.



	id	name 2	relation 1
1	3	Dylan	brother
2	6	Jack	brother
3	1	Harry	father
4	2	Chloe	mother
5	5	Alice	sister
6	4	Emily	sister

To restore the initial "unsorted" state for the table, click and select Reset View.

To reorder columns, use drag-and-drop for the corresponding cells in the header row. To restore the initial order of columns, click and select Reset View.



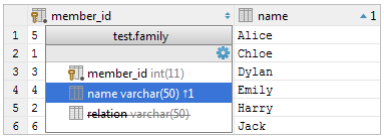
	member_id	relation	name 1
1	5	sister	lice
2	1	mother	hloe
3	3	brother	ylan
4	4	sister	mily
5	2	father	arry

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
   - Right-click any of the cells in the header row and select Column List.
   - Press `Ctrl+F12`.

   In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

To show all the columns, click and select Reset View.

See also, Using the Structure view to sort data, and hide and show columns.

Parameters pane

The Parameters pane shows the parameters detected in the input pane and lets you edit their values. To open or close this pane, use on the toolbar.

Parameter values can be specified just as text or numbers, or as Groovy expressions that contain object references and method calls. For example, the value for the `date` parameter in the query

```
SELECT o
                    FROM Order o
                    WHERE o.date >
                    :date
```

could be specified as

```
new java.sql.Date(System.currentTimeMillis() - 24*3600*1000)
```

To start editing a value, switch to the corresponding table cell and start typing. To indicate that you have finished editing a value, press `Enter` or switch to a different cell. To quit the editing mode and restore an initial value, press `Escape`.

When you select a row in the table, the corresponding parameter is highlighted in the input pane.

JSF Tool Window

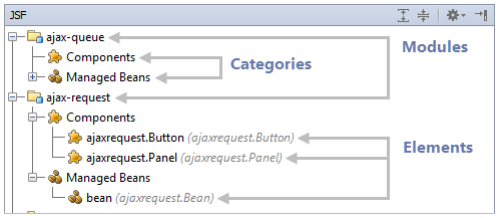
To open this tool window:

View | Tool Windows | JSF

See also, Showing a tool window.

Note that this tool window is available only if there is a library that implements JSF in the dependencies of one or more of your modules .

The JSF tool window provides a categorized hierarchical view of your JSF resources.



At the top of the hierarchy are your modules. One level below are categories.

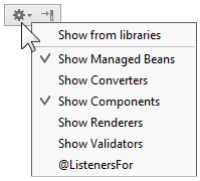
The categories correspond to JSF element types (managed beans, converters, components, etc.).

You can show or hide the categories. You can also open the elements shown in the tool window in the editor.

- Showing and hiding categories
- Opening elements in the editor

Showing and hiding categories

To show or hide a category, click on the title bar, and then click the necessary Show option.



The Show from libraries option is for showing the library elements referenced in your code.

Opening elements in the editor

You can open the elements shown in the JSF tool window in the editor. To do that, select the element of interest and press F4.

To open the elements which are the "leaves" of the tree (i. e. the ones at the bottom of the hierarchy), you can also use a double click.

JSTestDriver Server Tool Window

View | Tool Windows | JSTestDriver Server

- Start a local server link.

In this tool window, start and stop the JSTestDriver Server to run unit tests against and capture the browser to execute unit tests in.

Prerequisites

The tool window becomes available when the following prerequisites are met:

1. The JSTestDriver plugin is downloaded, installed, and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. The current project contains at least one test runner configuration files .

Options

Item	Tooltip and shortcut	Description
	Run a local server	Click this button to have IntelliJ IDEA launch the default JSTestDriver server.
	Stop the local server	Click this button to have IntelliJ IDEA stop the currently running JSTestDriver server.
Capture a browser using the URL		This read-only field shows the URL address to access the Remote Console of the JSTestDriver. Copy the URL address and open it in the browser of your choice.
		The icons indicate available browsers. The icon that corresponds to the browser you just opened, is active. Click the icon to get ready for executing tests.

Maven Projects Tool Window

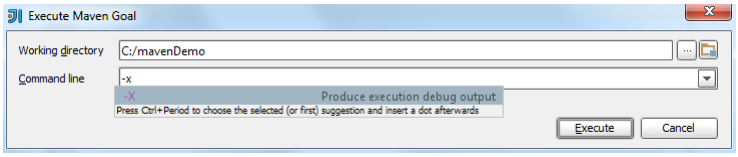
View | Tool Windows | Maven Projects

Use the Maven Projects tool window to view the available Maven projects, download sources and Javadocs, and execute phases of the build lifecycle. The tool window displays nodes for each Maven project, with the Lifecycle and Plugins subnodes. If at least one of the `pom.xml` files contains profile definition, the Profiles node is added to the tool window. This node contains all profiles defined in the Maven projects. The Maven Projects tool window also displays the Dependencies node if dependencies are added to your project.

In this section:

- Toolbar Buttons
- Context Menu Commands of a Lifecycle Phase
- Context Menu Commands of a Maven Project

Toolbar Buttons

Item	Description
	Click this button to synchronize all Maven projects with the IntelliJ IDEA project. See Importing tab of the Maven Integration dialog box.
	Click this button to launch Maven goals for generating sources and resources for the source and test directories, and read the resulting directory structure. According to the results of such generation, the IntelliJ IDEA folders are properly marked as the source or test roots . See import settings .
	Click this button to download missing sources and documentation. Select the desired download option from the submenu. You can set up automatic downloading of sources and documentation at the Importing page of the Maven Integration dialog.
	Click this button to add a Maven project. Select the desired `pom.xml` file in the dialog that opens .
	Click this button to execute the selected phase of the build lifecycle or a plugin goal. If several goals are selected, they will be executed in the same order as in the tree. Note that by default this button is disabled, to activate it you need to select a build phase or a plugin goal to run.
	Click this button to execute a maven goal using a command line. 
	Click this button to toggle offline mode .
	Click this button to turn on the Maven option *Skip test mode*, and omit running unit tests.
	Click this button to show the dependencies of the current Maven project in a UML pop-up frame .
	Click this button to collapse all nodes under the selected Maven project.
	Click this button to configure the settings of the current Maven project in the Maven Integration dialog box.
	Click this button to show reference page.
	Click this button to show the menu of the show options: – Group Modules: select this option to group the nodes by directories. – Show Ignored Projects:when this option is selected, the ignored nodes are shown in the tool window with a strikethrough. Otherwise, the ignored nodes are hidden. – Show Basic Phases Only: when this option is selected, IntelliJ IDEA shows only the basic build phases; otherwise, the complete list of phases is shown. – Always Show ArtifactId: select this option to display the artifactId that is specified in the pom.xml of your Maven project. – Show version: when this option is selected, IntelliJ IDEA displays the version of your Maven project that is specified in the pom.xml. – Show Toolbar - select this option to show the toolbar for your Maven projects. – Pinned, Docked, Floating, Windowed, Split Mode – Group Tabs - deselect this option to see the views on separate tabs if more than one view is available in a tool window. – Move to - select this option to move the Maven Projects tool window to either top, left or right. – Resize - select this option to resize the Maven Projects tool window.

Context Menu Commands of a Lifecycle Phase

Command	Description
Create <project>[phase]	Choose this command to create run/debug configuration for the selected phase of a lifecycle.
Run <project>[phase]	Choose this command to run the selected phase of a lifecycle with the phase-specific run/debug configuration.
Debug <project>[phase]	Choose this command to debug the selected phase of a lifecycle with the phase-specific run/debug configuration.
Execute before Make, Execute after Make	Choose these commands to set the respective flags for the selected phase of a lifecycle. So doing, *Before Make* and *After Make* comments appear next to the name of the node.
Execute before Rebuild, Execute after Rebuild	Choose these commands to set the respective flags for the selected phase of a lifecycle. In this case *Before Rebuild* and *After Rebuild* comments appear next to the name of the node.
Execute before Run/Debug	Choose this command to specify a run/debug configuration, prior to which the selected phase of a lifecycle should be executed. So doing, *Before Run* comment appears next to the name of the node.
Assign Shortcut	Choose this command to associate the selected phase with a keyboard shortcut . So doing, the comment with the shortcut appears next to the name of the node.

Context Menu Commands of a Maven Project

This section describes only those context menu commands that are not available from the toolbar.

Command	Description
Ignore Project / Unignore Project	Choose the Ignore Projects command to ignore project in build, or, on the contrary, include in build the previously ignored project. Ignored projects are not imported into IntelliJ IDEA.
Remove Project	Choose this command to delete the selected Maven modules from the Maven structure. So doing you can opt to delete the corresponding Maven module from the IntelliJ IDEA project as well.
Create / Open 'settings.xml' Create / Open 'profiles.xml'	Choose this command to create 'settings.xml' or 'profile.xml', or open such file if it has already been created.
Show Effective POM	Choose this command to generate the effective POM as an XML for this build, with the active profiles and super POM factored in. The effective POM displays the following information: – the default project source folders structure – the output directory – plug-ins required – repositories – a reporting directory which Maven will be using while executing the desired goals
Jump to source	Open in the editor the `pom.xml` file for the selected Maven project.

Messages Tool Window

View | Tool Windows | Messages

`Alt+0`

IntelliJ IDEA parses the output and displays it in a convenient format in the Messages window, letting you navigate to the source of the problem whenever possible.

Note The Messages tool window is only available if there are actually some messages to show.

On this page:

– Toolbar Buttons

– Results Context Menu

Toolbar Buttons

Item	Tooltip and shortcut	Description
	Rerun `Ctrl+F5`	Click this button to rerun compilation. The button is not available in Dart projects, see Managing Dart Packages and Assets for details.
	Rerun Pub Command	Click this button to run the last executed Pub command. The button is available only in Dart projects, see Managing Dart Packages and Assets for details.
	Pause output	Click this button to pause the compilation process. This button is enabled when compilation is in progress. The button is not available in Dart projects, see Managing Dart Packages and Assets for details.
	Stop/Stop Pub Process	Click this button to terminate compilation or Pub process. This button is enabled when compilation or a process is in progress.
	Close `Ctrl+Shift+F4`	Click this button to terminate the process and close the console.
	Previous/Next Message `Ctrl+Alt+Up` `Ctrl+Alt+Down`	Navigate to the previous/next message. The buttons are not available in Dart projects, see Managing Dart Packages and Assets for details.
	Export to Text File `Alt+O`	Save the current console contents. In the Export dialog, specify the target file or copy information to the Clipboard. Before saving, you can also edit the information to be saved. The button is not available in Dart projects, see Managing Dart Packages and Assets for details.
	Toggle tree/text mode	If this button is pressed, the output is displayed as plain text. If this button is released, the output is displayed as a tree view. The button is not available in Dart projects, see Managing Dart Packages and Assets for details.
	Show all messages	When this button is pressed, the only error messages are displayed. The button is not available in Dart projects, see Managing Dart Packages and Assets for details.
	Expand all `Ctrl+NumPad Plus` Collapse all `Ctrl+NumPad -`	Use these buttons to have all nodes expanded or collapsed. The buttons are not available in Dart projects, see Managing Dart Packages and Assets for details.
	Hide warnings	When this button is pressed, the tree of messages shows error messages only. The button is not available in Dart projects, see Managing Dart Packages and Assets for details.
	Autoscroll to source	If this button is pressed, the file that contains the selected error automatically opens in the editor, with the caret at the appropriate line. The button is not available in Dart projects, see Managing Dart Packages and Assets for details.
	Compiler properties	Configure the compiler settings . The button is not available in Dart projects, see Managing Dart Packages and Assets for details.
	Help	Use this button to navigate to the help topic for the tool window. The button is not available in Dart projects, see Managing Dart

Packages and Assets for details.

Results Context Menu

The menu items are not available in Dart projects, see Managing Dart Packages and Assets for details.

Item	Shortcut	Description
Jump to source	F4	Navigate to the selected item in the editor.
Copy	Ctrl+C	Take the line at caret to the clipboard.
Exclude from Compile		Skip the erroneous line in the next compilation.
Exclude from Validation		Choose this command to omit this warning in the next compilation.

Module Dependencies Tool Window

This tool window is available every time you perform the Analyze Module Dependencies command and displays module dependencies as they are defined in the module settings. If any cyclic dependencies are encountered in the selected module, they are specially marked in the tool window. In this tool window, you can change direction of the dependencies, and perform more detailed analysis of the source code.

In this section:

– Toolbar Buttons

– Context menu commands

Toolbar Buttons

Item	Description
	Click this button to close the current tab.
	Click this button to open the Specify Dependency Analysis Scope dialog box and analyze dependencies .
	Click this button to change direction of the dependencies.
	Click this button to open the reference.

Context menu commands

Item	Shortcut	Description
Expand All	`Ctrl+NumPad Plus`	Fold or unfold all nodes.
Collapse All	`Ctrl+NumPad -`	
Open Module Settings	`F4` `Ctrl+Enter`	Open settings of the selected module in the Modules page of the Project Structure dialog.
Analyze Dependencies Analyze Backward Dependencies Analyze Cyclic Dependencies		Choose one of these commands to perform analysis of dependencies. In the Specify Dependency Analysis Scope dialog box, specify scope of analysis.

NPM Tool Window

Context menu of a package.json - Show npm Scripts

View | Tool Windows | npm

- the tool window can be accessed this way only after you have opened it using the Show npm Scripts command.

On this page:

– Accessing the npm Tool Window
– Building a Tree of npm Scripts
– Running npm Scripts
– Toolbar
– Context Menu of a Tree
– Context Menu of a Script

Accessing the npm Tool Window

The tool window is available only when:

1. The Node.js runtime environment is installed on your computer.
2. The NodeJS repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
3. At least one `package.json` file is available in the current project.

The tool window opens when you invoke npm by choosing Show npm Scripts on the context menu of a `package.json` in the Project tool window or of a `package.json` opened in the editor. Use the tool window to run npm scripts.

As soon as you invoke npm, it starts building a tree of scripts defined within the `scripts` property of the `package.json` file on which it was invoked. If you have several `package.json` files in your project, you can build a separate script tree for each of them and run scripts without abandoning the previously built trees. Each tree is shown under a separate node.

Building a Tree of npm Scripts

To build a tree of scripts, do one of the following:

– Select the required `package.json` file in the Project tool window and choose Show npm Scripts on the context menu of the selection.
– Open the required `package.json` file in the editor and choose Show npm Scripts on the context menu of the editor.
– If the npm tool window is already opened, click on the toolbar and choose the required `package.json` file from the list. IntelliJ IDEA adds a new node and builds a scripts tree under it. The title of the node shows the path to the `package.json` file according to which the tree is built.

Running npm Scripts

To run a script from the tree, do one of the following:

– Double click the required script in the tree.
– Select the required script and choose Run <script name> on the context menu of the selection.
– Select the required script and press `Enter`.

The tool window shows the npm script output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed script is displayed on the title bar of the tool window.

To run several scripts, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required scripts, then choose Run on the context menu of the selection.

Toolbar

Item	Tooltip	Description
	Add package.json	Click this button to have a tree of scripts for another `package.json` file built. Choose the required `package.json` file from the pop-up list. IntelliJ IDEA adds a new node and builds a tree of scripts under it it.
	Remove package.json	Click this button to remove the tree of scripts under the selected node.
	Reload scripts	Click this button to have the tree of scripts under the selected node re-built. You may need a tree re-built after updating the corresponding `package.json` file because npm does not apply changes to trees on the fly.

Context Menu of a Tree

Item	Description
npm Settings	Choose this menu item to open the npm Settings dialog box and view or edit the Node.js configuration
Jump to Source	Choose this menu item to open the `package.json` file for which the current tree is built.
Reload scripts	Choose this menu item to have the tree of scripts under the selected node re-built.
Copy Path	Choose this menu item to save the path to the `package.json` file according to which the current tree was built to the clipboard.
Remove package.json	Choose this menu item to remove the tree of scripts under the selected node.

Context Menu of a Script

Item	Description
Run <script name>	Choose this menu item to run the selected script.

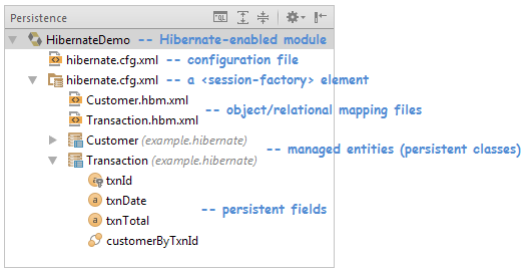
Edit <script name> settings	Choose this menu item to open the Run/Debug Configuration dialog box and edit the predefined settings for the selected script.
Jump to Source	Choose this menu item to open the `package.json` file for which the current tree is built and navigate to the definition of the selected script.

Persistence Tool Window

View | Tool Windows | Persistence

The Persistence tool window shows your JPA and Hibernate project items.

At the top hierarchical level are JPA- and Hibernate-enabled modules. One level below are configuration files, and <persistence-unit> and <session-factory> elements. Further down in the hierarchy are mapping files, persistent classes, etc.



The tool window lets you create new items such as configuration files, <persistence-unit> and <session-factory> elements, persistent classes and fields, navigate to related source code in the editor, open consoles and entity-relationship diagrams, and more.

Title bar icons

Item	Description
	Select a persistence unit or session factory (or any of the subordinate elements) and click this icon to open the JPA or Hibernate console . See also, Working with the JPA Console and Working with the Hibernate Console .
	Expand all the nodes.
	Collapse all the nodes.

Context menu commands

Item	Description
ER Diagram	For a persistence unit or session factory: open an entity-relationship diagram for the selected persistence unit or session factory.
Console (`Ctrl+Shift+F10`)	For a persistence unit or session factory: open the JPA or Hibernate console for the selected persistence unit or session factory. See also, Working with the JPA Console and Working with the Hibernate Console .
Assign Data Sources	For a persistence unit or session factory: associate the selected persistence unit or session factory with a database or DDL data source . See Associating persistence units and session factories with data sources.
Assign Naming Strategies	For a session factory: associate the selected session factory with a Naming Strategy implementation.
New (`Alt+Insert`)	Create a new item such as a persistent class, or configuration or mapping file, etc. See Using the New command.
Jump to Source (`F4`)	Open the selected file in the editor and switch to the corresponding code fragment.
Generate Persistence Mapping	Generate persistent classes and object/relational mappings for them. See Generating managed entity classes and O/R mappings.

Phing Build Tool Window

On this page:

- Accessing the Phing Build Tool Window
- Toolbar
- Context Menu

Accessing the Phing Build Tool Window

- Open a Phing build file in the editor or select it in the Project tool window, and then choose Add as Phing build file on the context menu of the selection.
- Choose View | Tool Windows | Phing Build on the main menu. The tool window can be accessed after you have opened it through the context menu of a Phing build file in the editor or in the Project tool window.

In this tool window:

- Specify the location of the `phing.bat` file.
- Configure a list of Phing build files for packaging, deploying, or testing PHP applications.
- Appoint targets to be executed before running or debugging according to specific configurations.
- Run entire build files and specific build targets.

The functionality of the Phing tool window is available through the toolbar buttons> and the context menu of a build file or target.

Toolbar

Item	Tooltip and shortcut	Description
+	Add	Click this button add a new script to the list. In the Select Phing Build Script dialog box, that opens, choose the desired file.
−	Remove	Click this button to remove the selected build file from the list.
▶	Run	Click this button to run the selected build file or target.
	Expand all `Ctrl+NumPad Plus`	Click this button to have IntelliJ IDEA show a tree of targets defined in the selected build file.
	Collapse all `Ctrl+NumPad -`	Click this button to have IntelliJ IDEA collapse all the targets in the selected build file.
	Properties	Click this button to have the Phing Properties dialog box opened. In this dialog box, specify the location of the `phing.bat` file. If necessary, configure the build procedure and the format of the output by specifying command line arguments in the Command Line Options text box. If the set of arguments is too large to fit in the text box, click the button or press `Shift+Enter` and type the desired arguments in the Command Line Options dialog box, that opens. The button is only available if the list of build files is not empty.

Context Menu

Item	Description	Available for
Run Build	Choose this option to have the selected build file executed.	Build files
Run Target	Choose this option to have the selected build target executed.	Build targets
Jump to Source	Choose this option to navigate to the source code of the selected build file or to the definition of the selected target.	Build files Build targets
Remove	Choose this option to remove the selected build file from the list.	Build files
Before Run/Debug	Choose this option to have the selected build target always executed before running or debugging the project according to specific run/debug configurations. In the Execute Target Before Run/Debug dialog box that opens, select the configurations before which you want the target executed.	Build targets
Assign shortcut	Choose this option to open the Keymap dialog box and configure a keyboard shortcut for the selected target as described in Configuring Keyboard Shortcuts .	Build targets
Mark to hide	Choose this option to make the selected target hidden and thus suppress showing it in the build file tree. This may be helpful when you have targets some that are only called by other ones and are never run alone. Hidden targets do not become visible when you expand the node of a specific build file or click Expand All button . To remove the hidden status of a target: 1. Select the build file in which it is defined and click the Settings button on the toolbar. 2. In the Phing Settings dialog box that opens, switch to the Hiding targets tab where the Hide check boxes next to the names of the hidden targets are selected. Clear the Hide check boxes next to the targets for which you want the status hidden removed.	Build targets
Properties	Click this button to have the Phing Properties dialog box opened. In this dialog box: - Specify the location of the `phing.bat` file. - If necessary, configure the build procedure and the format of the output by specifying command line arguments in the Command Line Options text box. If the set of arguments is too large to fit in the text box, click the button or press	Build files build

`Shift+Enter` and type the desired arguments in the Command Line Options dialog box, that opens.

- In the Hiding Targets tab, mark targets for hiding and remove the hidden status.
  Among the targets defined in a build file, you may have some that are only called by other targets and are never run alone. You can suppress showing such targets in the build file tree by marking them as hidden.

  Hidden targets do not become visible when you expand the node of a specific build file or click Expand All button .

The button is only available if the list of build files is not empty.

Phing Settings Dialog

IntelliJ IDEA editor | Context menu of a Phing build file | Add as Phing build file

Project Tool Window | Context menu of a Phing build file | Add as Phing build file

View | Tool Windows | Phing Build

The dialog box opens when you click the Settings button on the toolbar of the Phing Build tool window.

In this dialog box:

- Enable Phing integration in the current project.
- Configure Phing build properties externally to pass them through a command line.
- Show/hide targets in a specific file.

On this page:

- General Area
- Properties Tab
- Hiding Targets Tab

General Area

Item	Description
Path to Phing executable	In this text box, specify the location of the `phing.bat` file. Type the path manually or click the Browse button and choose the file location in the dialog box that opens.
Command line options	In this text box, specify the additional command line arguments to be passed to Phing when the build is launched. If necessary, click the and type the desired options in the Command Line Options dialog box. Type each option on a new line. When you close the dialog box, they are all displayed in the Command line options text box with spaces as separators.

Properties Tab

Use this tab to configure a list of property elements externally. These properties are passed to Phing through a command line when build execution is launched and therefore they are every time re-calculated dynamically.

Item	Description
Add	Click this button to have an empty line for a new property element added tot he list.
Remove	Click this button to remove the selected element from the list.
Property	In this text box, type the name of the property element.
Value	In this text box, specify the value of the element to be passed during build execution. Do one of the following: – Type the value manually. – To use IntelliJ IDEA macros , click the Insert macro button and configure a list of relevant macro definitions in the Macros dialog box that opens. To add a macro, select it in the list and click Insert.

Hiding Targets Tab

In this tab, specify which targets defined in the currently selected build file you want to have shown or hidden in the file tree view.

Among the targets defined in a build file, you may have some that are only called by other targets and are never run alone. You can suppress showing such targets in the build file tree by marking them as hidden.

Hidden targets do not become visible when you expand the node of a specific build file or click Expand All button .

Item	Description
Target	This field shows a list of all the targets defined in the current build file.
Hide	– When this check box next to the name of a target is selected, the target is not shown in the tree view of the file. – When this check box is cleared, the corresponding target is displayed in the tree view of the file.

You can mark a target as hidden both directly from the Phing Build tool window or from the Phing Settings dialog box. However, the hidden status can be removed only through the Phing Settings dialog box.

Problems Tool Window

This tool window is activated automatically if the Make project automatically option is enabled in Compiler settings . IntelliJ IDEA automatically compiles the project each time project files change on your disk (for example on save or autosave, or when you get the latest project revision from your version control system) and, if any problems are detected, they are displayed in the Problems tool window.

The Problems tool window only displays compiler errors that occurred as the result of the automatic make operation. On each automake, the list is updated with new errors, and old errors are deleted. Note that if automake is enabled, and you launch the make operation manually, the errors from this operation will also be displayed in the Problems tool window as well as in the Messages tool window.

Note The Problems tool window appears only if auto make option is enabled and your code contains a compilation error.

Toolbar

Item	Tooltip and shortcut	Description
	Previous message `Ctrl+Alt+Up`	Click this button to jump to the previous message.
	Next Message `Ctrl+Alt+Down`	Click this button to jump to the next message.
	Export to Text File `Alt+O`	Click this button if you want to export the contents of the tool window to a text file. Specify the name of the target file in the dialog that opens, and click Save.
	Help	Click this button to open IntelliJ IDEA help.
	Expand all `Ctrl+NumPad Plus` Collapse all `Ctrl+NumPad -`	Use these buttons to have all nodes expanded or collapsed.
	Autoscroll to Source	If this option is enabled, when you select a message in the Problems tool window, the focus in the editor automatically switches to the corresponding line in the source code.
	Compiler Properties	Click this button to open compiler settings .

Project Tool Window

View | Tool Windows | Project

`Alt+1`

The Project tool window lets you look at your project from various viewpoints and perform the tasks such as creating new items (directories, packages, files, classes, etc.), opening files in the editor, navigating to the code fragment of interest, and more.

Most of the functions in this tool window are accessed as context menu commands in the content pane and associated shortcuts.

- Views
- Context menu for the title bar
- Title bar icons
- Content pane
- Context menu commands for the content pane items
- File status highlights

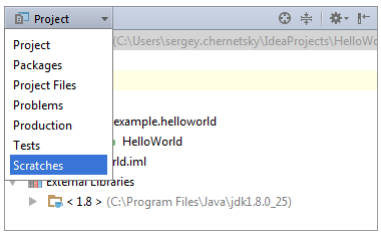
Views

The tool window provides a number of views.

Different views emphasize different project aspects and, generally, define which items are shown and how:

- Project view. In this view, all the project items along with their dependencies (SDKs and libraries) are shown. The emphasis is on the directory structure (though the packages are also shown).
- Packages view. The emphasis is on the package structure of the project. The modules, SDKs and libraries, by default, are not shown.
- Scopes views (Project Files, Problems, etc.). What is shown in the content pane is limited to the corresponding predefined or user-defined scope . In other respects, depending on the currently selected view options, a scope view may resemble the Project or the Packages view.
- Scratches view. This view lets you manage your scratch files and database consoles .

The necessary view is selected from the list in the left-hand part of the title bar or, if the views are represented by tabs, by clicking the corresponding tab.



To configure a view, use the corresponding options in the title bar context menu . The necessary options can also be accessed by clicking on the title bar.

Context menu for the title bar

The title bar context menu provides the options that let you control every aspect of the tool window appearance and behavior.

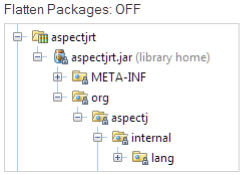
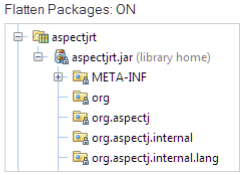
There are the options for configuring the project views . There are also the ones for controlling the tool window viewing modes .

The options that are on, have a check mark to the left of their names.

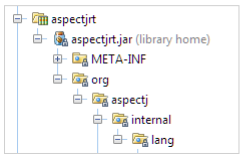
The menu also includes commands for switching between the views, resizing the tool window, and more.

To access the menu, right-click the list of views in the left-hand part of the title bar, or, if the views are represented by tabs, right-click the tab of interest.

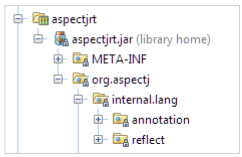
The following table lists and briefly explains the available commands and options.

Item	Shortcut	Description
Select Next View or Tab	`Alt+Right`	These are the commands for switching the views.
Select Previous View or Tab	`Alt+Left`	
Show List of Views or Tabs	`Alt+Down`	
Edit Scopes		Use this command to open the Scopes dialog in which you can create and edit used-defined scopes . Note that this command is available only if the current view is a scope view.
Flatten Packages		If this option is off, the packages are shown as a hierarchy. If this option is on, all the packages appear at the same level and are identified by their qualified names. Flatten Packages: OFF  Flatten Packages: ON 
Compact Empty Middle Packages or Hide Empty Middle Packages		If the Flatten Packages option is off, the option is called Compact Empty Middle Packages; if the Flatten Packages option is on, the option name is Hide Empty Middle Packages. This option lets you specify how or whether empty packages are to be shown. (Empty packages are ones that contain nothing but other packages.) If the option is on, empty packages are shown compacted or not shown at all (hidden).

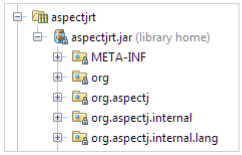
Compact Empty Middle Packages: OFF



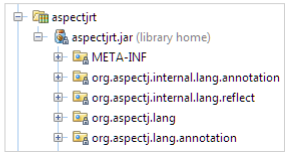
Compact Empty Middle Packages: ON



Hide Empty Middle Packages: OFF



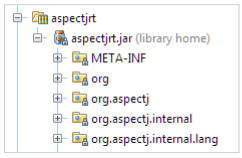
Hide Empty Middle Packages: ON



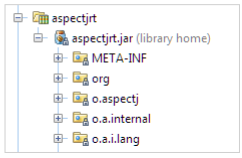
Abbreviate Qualified Package Names

This option is available only if the Flatten Packages option is on.
If the option is on, most of the initial `<name>.` fragments in qualified package names are abbreviated.

Abbreviate Qualified Names: OFF



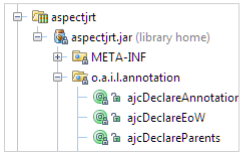
Abbreviate Qualified Names: ON



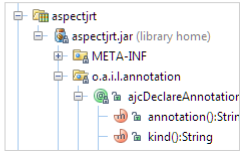
Show Members

If this option is on, the files in the tree that contain classes turn into nodes. When such node is unfolded, the contained classes with their fields, methods, and other members of the selected item are shown.

Show Members: OFF



Show Members: ON



Autoscroll to Source

If this option is on, IntelliJ IDEA automatically navigates from a file (or a class member) selected in the Project tool window to the corresponding source file (or its fragment) in the editor. If the corresponding file is not currently open, it will open automatically.

Autoscroll from Source

If this option is on, IntelliJ IDEA automatically navigates from a file in the editor to the corresponding node (file, class, field, method, etc.) in the Project tool window.

Sort by Type

If the option is off, the items (files, classes, etc.) are sorted alphabetically. If the option is on, the files are sorted by their extensions. The `.java` files appear in the following order: interfaces, classes, enumerations, etc.

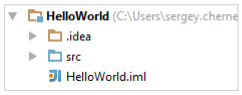
Folders Always on Top

If the option is on, all the folders are shown before the files. Otherwise, all the items are sorted alphabetically, and the files and folders appear intermixed.

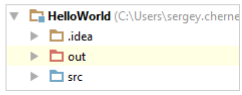
Show Excluded Files

This option is available only in the Project view. (In other views, excluded files are never shown.)
Turn this option on or off to show or hide excluded folders and files .

Show Excluded Files: OFF



Show Excluded Files: ON



Show Modules

This option is available only in the Packages and scope views. (In the Project view, modules are always shown.)
Turn this option on or off to show or hide modules.

Show Libraries Contents

This option is available only in the Packages view. (In the Project view, libraries are always shown; in the scope views, libraries are never shown.)
Turn this option on or off to show or hide libraries and their contents.

Note that within the Libraries category are the libraries included in module dependencies and also the SDKs associated with your modules.

Pinned, Docked, Floating, Windowed, Split Mode

These options let you control general appearance and behavior of the tool window. See Viewing Modes .

Remove from Sidebar

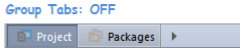
This command hides the tool window, removes the associated tool window button from the tool window bar and removes the tool window from the quick access menu (or).
To open the tool window again (and restore the associated features), use the main menu: View | Tool Windows | <Window Name>.

Group Tabs

If this option is on, there is a list in the left-hand part of the title bar from which you can select the necessary view. If this option is off, the views are represented by tabs which appear in the left-hand part of the title bar.

Group Tabs: ON

Group Tabs: OFF



Move to

To associate the tool window with a different tool window bar , select this command, and then select the destination tool window bar (Top, Left, Bottom or Right).

Resize

To resize the tool window by moving one of its borders, select this command, and then select the necessary Stretch to option.
Note that this command is not available for the floating mode.

Hide `Shift+Escape`

Use this command to hide the tool window.

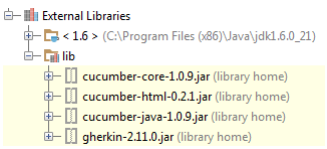
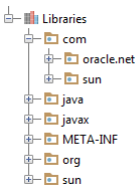
Title bar icons

Item	Shortcut	Description
	`Alt+Right` `Alt+Left`	If the views are currently shown as tabs (the Group Tabs option is off), this button appears to the right of the last visible tab. If the first or the last of the available views is currently selected, this button is shown as or . Click this button to open the list of views, for example, to select a different view.
		Click this icon to navigate from a file in the editor to the corresponding node (file, class, field, method, etc.) in the Project tool window. This icon is not available if the Autoscroll from Source option is currently on.
	`Ctrl+NumPad -`	Use this icon or shortcut to collapse all the nodes.
		Click this button to open the menu for configuring the current view and changing the tool window viewing modes . Note that most of the menu items are options that you can turn on or off. An option which is on has a check mark to the left of its name. The available options are a subset of the title bar context menu items. Depending on the current view, the menu may include the following options: – Edit Scopes – Flatten Packages – Compact or Hide Empty Middle Packages – Abbreviate Qualified Package Names – Show Members – Autoscroll to Source – Autoscroll from Source – Sort by Type – Folders Always on Top – Show Excluded Files – Show Modules – Show Libraries Contents – Pinned Mode, Docked Mode, Floating Mode, Windowed Mode, Split Mode – Remove from Sidebar – Group Tabs – Move to – Resize
	`Shift+Escape`	Use this icon or shortcut to hide the tool window. When used in combination with the Alt key, clicking this icon hides all the tool windows attached to the same tool window bar .

Content pane

The content pane shows the project items such as modules, packages, directories, files, classes, libraries, etc.

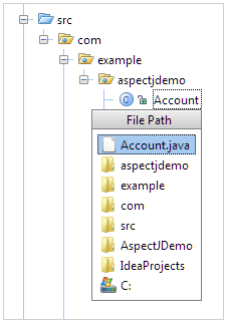
The icons for the main categories (node types) are shown and briefly explained in the following table. The icons used for the main file types are listed in File Types Recognized by IntelliJ IDEA ; the icons for the main symbols (classes, fields, methods, etc.) are shown in Symbols .

Item	Description
	A module.
	A package.
	A folder (directory). Different folder types have different colors: – An "ordinary" folder . – A source folder . – A test source folder . For more information on folder types, see Content Root .
	Libraries, a category for grouping the SDKs associated with your modules and the libraries included in module dependencies. When the Project view is selected, there is one such node for the whole project which is labeled External Libraries. The subcategories in this case are the SDKs (for example, JSDKs) and individual libraries ( ).  When the Packages view is selected, the corresponding node or nodes are labeled Libraries. If the Show Modules and the Show Libraries Contents options are selected, there is an individual Libraries node for each of the modules. The main subcategory in this case is a package ( ). 

Context menu commands for the content pane items

When you right-click an item in the content pane, the context menu for this item is shown. This menu provides access to all the functions available for the selected item.

The following table lists and briefly explains the most frequently used context menu commands.

Item	Shortcut	Description
New	`Alt+Insert`	Use this command to create a new item (module, package, directory, file, class, etc.) within the selected one (project, module, directory or package) In the Scratches view, this command also lets you create a database console.
Add Framework Support		For a module: use this command to add support for certain frameworks and technologies in the selected module. (The Add Frameworks Support Dialog will open.)
Cut	`Ctrl+X`	Use this command to move the selected item or items from the current location to the clipboard.
Copy	`Ctrl+C`	Use this command to copy the selected item or items to the clipboard.
Copy Path(s)	`Ctrl+Shift+C`	Use this command to copy the full path(s) of the selected item or items to the clipboard.
Copy Reference	`Ctrl+Shift+Alt+C`	Use this command to copy a reference to the selected item to the clipboard.
Paste	`Ctrl+V`	Use this command to insert the contents of the clipboard into the selected location.
Jump to Source	`F4`	Use this command to open the selected file in the editor. If the file is already open, the corresponding editor tab will become active.
Open Module Settings	`F4`	Use this command to see the settings for the selected module. These will be shown on the Modules page in the Project Structure dialog .
Find Usages	`Alt+F7`	Use this command to find the usages of the selected item. (The Find Usages dialog will open.)
Find in Path	`Ctrl+Shift+F`	Use this command to perform a text search. (The Find in Path dialog will open.)
Replace in Path	`Ctrl+Shift+R`	Use this command to perform text search-and-replace. (The Replace in Path dialog will open.)
Analyze		Use this command to access the functions related to code inspection and analysis .
Refactor		Use this command to perform one of the refactorings available for the selected item.
Add to Favorites		Use this command to add the selected item to an existing or new list of favorite items. See Managing Your Project Favorites .
Show Thumbnails	`Ctrl+Shift+T`	Use this command to view thumbnails for image files located in the selected directory. (The Thumbnails tool window will open.)
Browse Type Hierarchy	`Ctrl+H`	For a file (normally, a class): use this command to see the class hierarchy for the selected file (class). (The Hierarchy tool window will open.) See also, Viewing Structure and Hierarchy of the Source Code .
Reformat Code	`Ctrl+Alt+L`	Use this command to reformat the source code in the selected file or in all files in the current directory. (The Reformat Code dialog will open.) See also, Reformatting Source Code .
Optimize Imports	`Ctrl+Alt+O`	Use this command to optimize imports (i.e. to remove unnecessary `import` statements) in the selected file or in all files in the current directory. (The Optimize Imports dialog will open.) See also, Optimizing Imports .
Delete	`Delete`	Use this command to delete the selected item. Use with care!
Change Dialect (<CurrentDialect>)		For SQL files and database consoles: change the SQL dialect associated with the file or console.
Remove Module	`Delete`	Use this command to remove a module from your project. Note that the files that make up the module are not physically removed from the disk.
Make Module '<name>'		Use this command to make the current module. See Compilation Types and Reviewing Compilation and Build Results .
Compile '<name>'	`Ctrl+Shift+F9`	Use this command to compile the selected source file or all the source files in the selected directory. See Compilation Types and Reviewing Compilation and Build Results .
Run '<item_name>'	`Ctrl+Shift+F10`	For an SQL file or database console: execute all the statements contained in the selected file or console.
Local History		Use this command to view local history for the selected file or directory, or to create a label for the current version of your project. See Local History and Using Local History .
Synchronize '<item_name>'		Use this command to synchronize the selected item with its version saved in the file system. (If you change a file or directory contents externally, IntelliJ IDEA, under certain circumstances, may not be aware of the corresponding changes unless you use this command.)
Show in Explorer		Use this command to open a file browser (e.g. Windows Explorer or Finder) and show the selected item there.
File Path	`Ctrl+Alt+F12`	Use this command to open the File Path menu. This menu shows the path from the file system root to the selected element with individual directories as the menu items.  When you select an item in this menu (e.g. a directory), a file browser (e.g. Windows Explorer or Finder) opens, and the selected item is shown there.
Compare With		Use this command to compare the selected file or directory with another file or directory. Select the other file or directory the

	Ctrl+D	dialog that opens . See Comparing Folders and Differences viewer for folders .
Compare File with Editor		Use this command to compare the selected file with the file open on an active editor tab. See Comparing Files and Differences viewer for files .
Mark Directory As		Use this command to make the selected directory a source root or a test source root , to make the directory excluded, etc. The necessary category for the directory is selected from the submenu. For more information on the categories for directories, see Content Root .
Mark as Plain Text		Use this command to exclude the selected file from project, so it is ignored by inspections, code completion, navigation, etc. The file will be indicated with a special icon and shown as plain text in the editor. For more details, see Excluding Files from Project .
Diagrams	Ctrl+Shift+Alt+U or Ctrl+Alt+U	Use this command to open a diagram (e.g. a UML diagram) for the selected item. For more information, see Diagram Reference .
Update Copyright		Use this command to update the copyright notice for the selected files and folders. See Configuring Copyright Profiles and Generating and Updating Copyright Notice .
WebServices		Use this command to access the functions related to developing Web services. See Web Services and Web Service Clients .

File status highlights

IntelliJ IDEA uses colors to denote VCS file status in the Project tool window. The following table presents information about the meaning of the colors.

Color	File Status	Description
Black	Up to date	File is unchanged. PhantomsTest.java
Gray	Deleted	File is scheduled for deletion from the repository. privatent.txt
Blue	Modified	File has changed since the last synchronization. Advice.java
Green	Added	File is scheduled for addition to the repository. ResultTest.java
Violet	Merged	File is merged by your VCS as a result of an update. VcsHistoryDialog.java
Brown	Unknown	File exists locally, but is not in the repository, and is not scheduled for adding. Test.xml
Olive	Ignored	File will be ignored in any VCS operation. Test.html
Light brown	Hijacked	File is modified without checkout . This status is valid for the files under Perforce, ClearCase and VSS. HelpTOC.xml
Red	Merged with conflicts	During the last update, file was merged with conflicts. HistoryTestCase.java
Lilac	Externally deleted	File is deleted locally, but was not scheduled for deletion, and still exists in the CVS repository. test1.txt
Dark cyan	Switched	The file is taken from a different branch than the whole project. This status is valid for CVS and SVN. test1.txt

REST Client Tool Window

Tools | Test RESTful Web Service

View | Tool Windows | REST Client

Use the REST Client tool window for testing a RESTful Web Service . The tool window is intended for composing and submitting test requests to Web service methods based on the service API, as well as for viewing and analyzing server responses.

Please note the following:

- View | Tool Windows | REST Client - the tool window can be accessed this way after you have opened it using the Tools | Test RESTful Web Service command.
- The tool window is available only when the REST Client bundled plugin is enabled. The plugin is active by default. If not, activate it in the Plugins page of the Settings dialog box.

IntelliJ IDEA supports integration between the source code and the contents of the REST Client tool window controls.

On this page:

- Common Request Settings
  - Toolbar
- Request Tab
- Cookies Tab
- Response Tab
- Response Headers Tab

Common Request Settings

In this area, choose the request method and specify the data to compose the request URI from.

The server response code and the content length are shown in the upper-right corner of the REST Client tool window.

Item	Description
HTTP method	In this drop-down list, specify the request method . The available options are: - GET - POST - PUT - PATCH - DELETE - HEAD - OPTIONS
Host/port	In this text box, type the URL address of the host where the target Web service is deployed and the port it listens to. By default, the port number is `80` . If another port is used, specify it explicitly in the format `<host URL>:<port number>` .
Path	In this drop-down list, specify the relative path to the target method.

You can enter the entire URL address of a method to test in the Host/port text box. Regardless of the chosen HTTP method, upon pressing ▶ IntelliJ IDEA will split the URL address into the host/port and the path to the method. The extracted relative path will be shown in the Path text box and the extracted parameters will be added to the list in the Request Parameters pane of the Request tab.

Toolbar

Item	Tooltip and shortcut	Description
▶	Submit Request	Click this button to submit the generated test request to the server. **Note** If a server is not trusted, IntelliJ IDEA shows a dialog box suggesting you to accept the server, or reject it. If you accept the server as trusted, IntelliJ IDEA writes its certificate to the trust store. On the next connect to the server, this dialog box will not be shown.
	Replay Recent Requests	Click this button to have a Recent Requests pop-up list displayed and select the relevant request. The fields are filled in with the settings of the selected request. Click the Submit Request button ▶.
	Export Request	Click this button to have the current request settings saved in an XML file so they are available in another IntelliJ IDEA session. In the dialog box that opens, specify the name of the file to save the settings in and its parent folder. When necessary, you can retrieve the saved settings and run the request again.
	Import Request	Click this button to have the settings of a previously saved request retrieved from an XML file. In the dialog box that opens, select the relevant XML file.
	Generate Authorization Header	Click this button to open the Generate Authorization Header dialog box and specify your user name and password for accessing the target RESTful Web service through. Based on these credentials IntelliJ IDEA will generate an authentication header which will be used in basic authentication . Learn more at http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm .
	Update resource paths from code	Click this button to synchronize the contents of the Path drop-down list with the @Path annotations.
	Configure HTTP Proxy	Click this button to specify proxy server settings in the dialog box that opens.
	Close	Close the REST Client tool window.

Help Show this page.

Request Tab

Use this tab to specify the parameters to be passed to the service in the generated test request either through the query string for `GET` requests or through the request body for other request types. Also configure interaction between the client side and the Web service by specifying the format of data that the service and the client accept.

Item	Description
Headers	In this pane, specify the technical data included in the request header. These data are passed through header fields and define the format of the input parameters (accept field), the response format (content-type field), the caching mechanism (cache-control field), etc. To add a field to the list, click Add, then specify the field name in the Name text box and the field value in the Value drop-down list. The set of fields and their values should comply with the Web service API. In other words, the specified input format should be exactly the one expected by the Web service as well as the expected response format should be exactly the one that the service returns. For `accept`, `content-type`, and some other fields IntelliJ IDEA provides a list of suggested values. Choose the relevant format type from the Value drop-down list.
Request Parameters	In this pane, specify the parameters to be passed to the target method through a query string inside the URL. This approach is used for requests of the type `GET`. By default, the pane shows an empty list with one line. – To add a parameter, click Add, then specify the name of the parameter in the Name text box and the value of the parameter in the Value drop-down list. – To delete a parameter from the list, select it and click Remove. – To suppress sending the specified query string parameters and disable the controls in the Request Parameters pane, press the Don't send anything toggle button. The set of parameters and their types should comply with the Web service API, in particular, they should be exactly the same as the input parameters of the target method. – select this check box This may be helpful, for example, if you want to test passing parameters through other request methods in the request body but still preserve the data typed in the Query string parameters pane.
Request Body	The pane is disabled when the `GET`, `DELETE`, `HEAD`, or `OPTIONS` request method is selected. In this pane, specify the input parameters to be passed to the target method inside a request message body. – Empty: choose this option to send a request with an empty body. – Text: choose this option to send a request with a string of parameters with values. Specify the parameters in the text box next to the option. – File contents: choose this option to have the parameters inserted in the request body from a local text file. Specify the source text file in the File to send field. – File upload(multipart/form-data): choose this option if you want to pass a binary file as a parameter, which requires that the file be converted. Specify the source binary file in the File to send field.

Cookies Tab

Use this tab to create, save, edit, and remove cookies, both received through responses and created manually. The name and value of a cookie is automatically included in each request to the URL address that matches the domain and path specified for the cookie, provided that the expiry date has not been reached.

The tab shows a list of all currently available cookies that you received through responses or created manually. The cookies are shown in the order they were added to the list. When you click a cookie, its details become editable and are displayed in text boxes.

Item	Description
Name	In this fields, specify the name of the cookie to be included in the request.
Value	In this field, specify the value of the cookie to be included in requests.
Domain	In this field, specify the host and port the requests to which must be supplied with the name and value of the cookie.
Path	In this field, specify the path of the URL the requests to which must be supplied with the name and value of the cookie.
Expiry date	In this field, specify the expiry date of the cookie.
+	Click this button to add a new row to the list and define a new coolie in it.
–	Click this button to remove the selected cookie from the list.

Response Tab

Use this tab to view responses from the Web service. By default, responses are shown in the plain text form. Use the icons of the tab to have them displayed in the editor in the HTML, XML, and JSON formats.

Item	Tooltip	Description
	View as HTML	Click this button to open a new tab in the main editor window and display there the response as HTML.
	View as XML	Click this button to open a new tab in the main editor window and display there the server response as XML.
	View as JSON	Click this button to open a new tab in the main editor window and display there the server response as JSON.
	Open in browser	Click this button to view the response in your default Web browser.

Response Headers Tab

The tab shows the technical data provided in the headers of Web service responses.

Remote Host Tool Window

Tools | Deployment | Browse Remote Host

View | Tool Windows | Remote Host

Use this tool window to view the folder structure of the target FTP/FTPS/SFTP servers and the data uploaded to them.

Getting Access to the Remote Host Tool Window

- View | Tool Windows | Remote Host - the tool window can be accessed this way after you have opened it using the Tools | Deployment | Browse Remote Host command.
- The tool window is available only when the Remote Hosts Access bundled plugin is enabled. The plugin is active by default. If not, activate it in the Plugins page of the Settings dialog box.

Toolbar

Item	Tooltip and Shortcut	Description
Remote host		From this drop-down list, select the desired remote host configuration.
	Shift+Enter	Click the browse button to add a new server .
	Collapse All Ctrl+NumPad -	Click this button to have all nodes in the view collapsed.
	Refresh Ctrl+F5	Click this button to refresh the view.
	Close Ctrl+Shift+F4	Click this button to close the tool window.
	Help F1	Click this button to show this reference page.

Context Menu

Item	Description
Upload Here	Choose this option to have the files from the currently opened project uploaded according to the selected configuration. If the action is invoked from the context menu of a file, the corresponding local file is uploaded. If the action is invoked from a folder, the entire folder is uploaded.
Download from here	Choose this option to download the selected file or folder to the currently opened. The existing local files will be updated and the missing files will be created.
New	Choose this option to create a new remote file or folder in the selected folder. The option is available only from the context menu of a folder.
Rename	Choose this option to rename the selected file or folder.
Delete	Choose this option to remove the selected file or folder.
Cut	Choose this option to copy the name of the selected file or folder to the clipboard and remove the file or folder from the tree.
Copy	Choose this option to copy the name of the selected file or folder to the clipboard.
Paste	Choose this option to insert the previously copied name of a file or a folder into the tree.
Edit Local File	When you select a file and choose this option, IntelliJ IDEA opens its local copy in the editor and moves the focus to the corresponding. The option is available only when the project with the corresponding local file is currently opened i the editor.
Sync with Local	Choose this option to compare the selected remote folder with its local version. In the Differences Viewer for Folders that opens, explore the differences and synchronize the files , where applicable. See Comparing Deployed Files and Folders with Their Local Versions for details.
Compare with Local Version	Choose this option to compare the selected remote file with its local version. In the Differences Viewer for Files dialog box, that opens, explore the differences and apply them, if necessary, using the » and « buttons. For details, see Viewing Differences Between Files . See Comparing Deployed Files and Folders with Their Local Versions for details.
Edit Remote File	Choose this option to edit the selected file in the IntelliJ IDEA editor without adding it to the currently opened project. See Editing Individual Files on Remote Hosts for details.
Copy Path	
Exclude Path	Choose this option to exclude the selected folder from upload/download, see Excluding Files and Folders from Upload/Download for details.
Copy Path	Choose this option to copy the absolute path to the selected file or folder on the server to the clipboard.

Run Tool Window

Run | Run

View | Tool Windows | Run

`Alt+4`

The Run tool window displays output generated by your application. If you are running multiple applications, each one displays in a tab named after the run/debug configuration applied.

The appearance of each tab depends on the type of the application being run and can include additional toolboxes and panes.

- Run Tool Window
  - Run Toolbar
  - Context menu
  - Console Toolbar
  - Karma Server tab
- Deployment Console
- Logs Tab
- Test Runner Tab
- Export Test Results
- Groovy Shell
- Interactive Groovy Console
- Play Framework (Play Console)

The main toolbar of the Run tool window lets you rerun, stop, pause, or terminate an application. The following table contains descriptions of the buttons that are common for most applications.

Run Toolbar

Item	Tooltip and shortcut	Description
	Rerun `Ctrl+F5`	Click this button to stop the current application and run it again. When an application is stopped ( ), this button toggles to .
	Rerun `Ctrl+F5`	Click this button to rerun the current application. This button appears, when an application is stopped ( ). When an application is running, this button toggles to .
	Pause Output	Click this button to have the process output paused.
	Dump Threads `Ctrl+Break`	Click this button to dump all threads of the current process showing their status in the Sun format.
	Exit	Click this button to terminate the current process gracefully using in-process internal mechanisms.
	Stop `Ctrl+F2`	Click this button to terminate the current process externally by means of the standard `shutdown` script.
	Restore Layout	Click this button to to have the changes to the current layout abandoned and return to the default state.
	Pin	Use to pin or unpin the tab. If a tab is pinned, the results for the next command are shown on a new tab.
	Close `Ctrl+Shift+F4`	Click this button to close the selected tab of the Run tool window and terminate the current process.
	Help `F1`	Use this icon or shortcut to open the corresponding help page.

Context menu

Item	Description

Console Toolbar

Item	Tooltip and shortcut	Description

	Up/down the Stack Trace
	`Ctrl+Alt+Up`
	`Ctrl+Alt+Down`
		Click this button to navigate up or down in the stack trace and have the cursor jump to the corresponding location in the source code.

	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the

		cursor jump to the corresponding location in the source code.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.
	Toggle tasks executions/text mode	Click this button to view Gradle task execution in a tree mode.

Karma Server tab

The tab is shown only when you run JavaScript unit tests using the Karma test runner. Use the tab to view and analyze information from the server. For details, see Running Unit Tests on Karma .

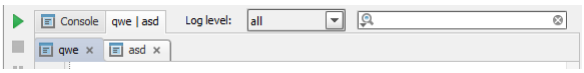
Deployment Console

This console shows which of the configured and marked for deployment artifacts are successfully deployed and which are not. Use the console to deploy and undeploy artifacts and configure their execution on the server.

Item	Tooltip and Shortcut	Description
	Artifact is deployed successfully	This icon next to an artifact indicates that the artifact has been successfully deployed to the server.
	Artifact is not deployed	This icon next to an artifact indicates that the artifact has not been deployed to the server yet or has been undeployed from the server.
	Deploy All	Click this button to have IntelliJ IDEA deploy all the artifacts from the list of items to be deployed in the Deployment tab of the Run/Debug Configuration dialog box.
	Undeploy	Click this button to have the selected artifact undeployed from the server.
	`Ctrl+F5` Refresh Deployment Status	Click this button to synchronize the deployment status indications for artifacts with the server.
	Update Resources On Frame Deactivation	Click this button if you want the application to be updated automatically when you switch from IntelliJ IDEA to a different application. (Switching to a different application is referred to as frame deactivation.) This button is a toggle that turns the corresponding option on or off. If enabled, the application assets to be updated are defined by the current setting of the On frame deactivation option in the active run/debug configuration . Note that by changing the state of the button you also change the setting of On frame deactivation in the current run/debug configuration. For example, if you disable the button, the setting changes to Do nothing.

Logs Tab

The Logs tab is available, if logging has been activated in the current Run/Debug configuration. This tab is named after the log file. If there are multiple log files enabled for a run/debug configuration, the tab's name is composed of all the log file names delimited with the "|" characters; so doing, each specific log is displayed in its own tab:



Log level

The logs can contain messages of the following types:

- All
- information
- warnings
- errors

Use the drop-down list of available message types to filter the log output and navigate through the messages.

Search

Use the text field to type a string to find, and press `Enter` to start the search.

Click to reveal to history of searches.

Click to clear the text field.

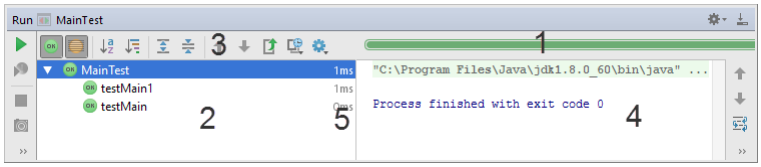
Test Runner Tab

In this section:

- Basics
- Run toolbar
- Testing toolbar
- Test status icons
- Output pane
- Context menu commands

Basics

The Test Runner tab opens in the Run tool window when a testing session begins, and features the same toolbar buttons .



1. The color of the status bar indicates whether the tests have passed successfully. If at least one of the tests fails, the status bar turns red.
2. The left-hand pane shows the tree view of all tests within the current run/debug configuration.
   - The root node represents the test case selected to run.
   - The nested nodes represent the hierarchy of test suites and test cases.
   - The leaf nodes represent the individual tests.

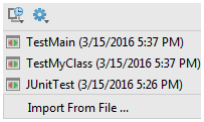
   The status of each test is indicated by an icon . Double-click a node to open the respective test class or test method in the editor.
3. The testing toolbar provides controls that enable you to monitor the tests and analyze results. Some of the commands are duplicated on the context menus of the test tree nodes.
4. The Output pane shows the output of the current test suit.
5. The inline statistics show the list of executed tests with the execution time of each test.

Run toolbar

Item	Tooltip and Shortcut	Description
	Rerun Ctrl+F5	Click this button to rerun the current process. The process reruns always in the same console regardless of whether this console is pinned or not.
	Rerun Failed Tests	Click this button to have IntelliJ IDEA execute failed tests. If you press Shift and click this button, you can choose whether you want to Run the tests again, or Debug, i.e. rerun the failed tests in the Debug mode.
	Pause Output	Click this button to have the process output paused.
	Dump Threads Ctrl+Break	Click this button to dump all threads of the current process showing their status in the Sun format.
	Exit	Click this button to terminate the current process gracefully using in-process internal mechanisms. This button is available for GWT applications only.
	Stop Ctrl+F2	Click this button to terminate the current process externally by means of the standard mechanisms.
	Layout	Click this button to choose a layout of the toolbars by selecting the desired option from the pop-up menu.
	Pin	When this button is pressed, the current tab will not be overwritten; instead, the results of the next command will be displayed in a new tab.
	Ctrl+Shift+F4	Click this button to close the selected tab of the Run tool window and terminate the current process.
	F1	Click this button to show reference.

Testing toolbar

Item	Tooltip and Shortcut	Description
	Show Passed	Click this button to show tests that passed successfully.
	Show Ignored	Click this button to show the ignored tests in the tree view of all tests within the current run/debug configuration or test class.
	Sort alphabetically	Click this button to sort tests in alphabetical order.
	Sort by duration	Click this button to sort tests by duration.

	Expand All/Collapse All Ctrl+NumPad Plus Ctrl+NumPad -	Click these buttons to have all nodes in the tree view of tests expanded. These buttons are only available if the tested application contains more than test case.
	Previous/Next Failed Test Ctrl+Alt+Up / Ctrl+Alt+Down	Click these buttons to navigate between the failed tests.
	Show coverage per test	Click this button to have the code coverage results shown in the Project tool window. The button is available after all the tests are executed provided that code coverage is enabled in the current run configuration.
	Export Test Results	Click this button to have the results of the selected test saved in a file. In the Export Test Results that opens, specify the file to save the output in and the format in which the data will be saved. If you want to view the test results later, choose the XML format. To view the saved test results later, click and specify the XML file where they are stored.
	Import Test Results	Click this button to view the test results that you previously saved in an XML file or the results that IntelliJ IDEA has kept in its internal history. The pop-up menu that opens shows a list of internally saved results of test sessions, each item is supplied with the name of the run configuration and a time stamp:  TestMain (3/15/2016 5:37 PM) TestMyClass (3/15/2016 5:37 PM) JUnitTest (3/15/2016 5:26 PM) Import From File ... – To view the results of a testing session from the IntelliJ IDEA history, select the item with the suitable run configuration and time stamp. – To load the previously exported results, choose Import from file and then choose the required XML file in the dialog box that opens. The loaded test results are shown in the tab and the name of the corresponding run configuration is displayed on the title bar. To re-run the tests from the loaded session, click .
		Click this cog button to access the context menu with the following options: – Track Running Test: turn this option on to monitor execution of the current test. If a test suite contains multiple tests, the tree view of tests expands to show sequential test methods, as they are executed. – Show Inline Statistics: turn this option on to have the statistics shown next to a test result, displaying the time used for executing each test. – Scroll to Stacktrace: turn this option on to have the console scroll to the beginning of the trace of the last failed test. If you click the root node (the test package) in the tree view with this option turned off, the console will show the very beginning of the test. This option is helpful when a test package contains multiple test classes and test methods. If some of the tests fail, you can scroll in the console to the beginning of a stack trace of an exception or assertion. – Open Source at Exception: use this option to explore the results of a test that fails as an error, throwing an uncaught exception. If you double-click the failed test class or method in the tree view with this option turned on, the respective test class or method will open in the editor, with the caret placed at the line that caused the problem. – Autoscroll to Source: turn this option on to have the currently selected test in the tree view synchronized with the editor automatically. – Set Auto Test Delay: point to this node to reveal the list of available delays between file saving and rerunning tests. The selected value gets the check mark. – Select First Failed Test When Finished: turn this option on to have the first failed test automatically selected in the tree view upon completing the tests. – Include Non-Started Tests into Rerun Failed

Test status icons

Icon	Description
	Test error. This status is assigned to tests that caused an exception from the tested source code.
	Test failed. If at least one test receives this status, the entire tree is marked as terminated as well.
	Test ignored. In the PHP context, this icon indicates a skipped test.
	Test in progress.
	Test passed successfully.
	Test paused.
	Test terminated. This status is assigned to tests that were cancelled by clicking the Stop button . If at least one test receives this status, the entire tree is marked as terminated as well.
	Test not run. This status is assigned to a test when IntelliJ IDEA has not launched it at all, for example, because testing was terminated by clicking the Stop button .

Output pane

This pane shows output of each test, generated at runtime, including all the messages sent to the output stream, and the error messages. The following table shows the toolbar buttons and context menu commands available for the Output pane.

Item	Keyboard Shortcut	Description
	Up the Stack Trace	Click this button to navigate up in the stack trace and have the cursor jump to the corresponding location in the source code.
	Ctrl+Alt+Down	

	Down the Stack Trace Ctrl+Alt+Up	Click this button to navigate down in the stack trace and have the cursor jump to the corresponding location in the source code.
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location in the source code.
	Print	Click this button to configure printing out the console output in the Print dialog box that opens.
Clear All		Choose this item on the context menu to have all messages for the selected test deleted.
Copy Content	Ctrl+C	Choose this item on the context menu to have the current contents of the Output pane placed to the Clipboard.
Compare with Clipboard		Choose this item on the context menu to invoke the Differences viewer for files which shows the current contents of the Clipboard in the left-hand pane and the contents of the Output pane for the selected test in the right-hand pane.

Context menu commands

Command	Keyboard shortcut	Description
View assertEquals Difference	Alt+Enter	Choose this command to show the Differences viewer for the strings being compared.. This command is only available when assertion has failed.
Run <test name>	Ctrl+Shift+F10	Run the selected test with the current temporary run/debug configuration or choose another configuration from the subordinate context menu.
Debug <test name>		Debug the selected test with the current temporary run/debug configuration or choose another configuration from the subordinate context menu.
Run <test name> with Coverage	Ctrl+Shift+F10	Run the selected test and collect coverage data in accordance with the current temporary run/debug configuration .
Create <test name>		Create a run/debug configuration on the base of the selected test.
Save <test name>		Save the temporary run/debug configurationn
Jump to Source	F4	Choose this command to move the focus to the editor, to the definition of a test class, or test method.
Show Source	Ctrl+Enter	Choose this command to open source code in the editor, but leave the focus with the Test Runner tab.
Navigate to testdata		If a test class has a file with testdata, IntelliJ IDEA selects this file in the Project tool window.

Export Test Results

Run tool window | Test Runner

The dialog box opens when you click the Export Test Results button in the Test Runner tab of the Run tool window .

In this dialog box, choose the format in which you want the test output saved and the file to save the test results in.

Item	Description
Export format	In this area, choose the desired output format. The available options are: – HTML – XML – Custom, apply XSL template: choose this option to have the results presented according to your own code style. Select the relevant `*.xsl` code style definition file in the file chooser.
Output	In this section, specify the target file name (File name), and directory (Folder).
Open exported file in browser	Select this check box to automatically open the above defined file with the test results in the default browser.

Groovy Shell

Tools | Groovy Shell

A Groovy shell is a command-line application that lets you evaluate Groovy expressions, functions, define classes and run Groovy commands. The Groovy shell can be launched in Groovy projects.

Item	Tooltip and Shortcut	Description
	Stop `Ctrl+F2`	Click this button to stop the current process.
	Close `Ctrl+Shift+F4`	Click this button to close the selected tab of the Run tool window and terminate the current process.
	Execute Groovy Code (`Ctrl+Enter`)	Run code, entered in the console.
	Help `F1`	Use this icon or shortcut to open the corresponding help page.

Interactive Groovy Console

Tools | Groovy Console

Interactive Groovy console can be launched in any project.

Note that if dependencies in your project contain a Groovy library then the specified Groovy library will be used to launch the Groovy console. If the dependencies do not contain a Groovy library then the bundled Groovy library of the Groovy version 2.3.9 will be used.

Editor for Groovy Console

Use the editor for the interactive Groovy console to write and evaluate your code.

Item	Description
	Click this icon to run the code entered in the editor in the Groovy console. You can view the results in the run tool window for Groovy console.
	Click this icon to see the module in which the Groovy console was launched. It might be helpful for multi-module projects.

Run Tool Window for Groovy Console

Use this tool window to view the results of the Groovy console execution.

Item	Tooltip and Shortcut	Description
	Stop Ctrl+F2	Click this button to stop the current process.
	Close Ctrl+Shift+F4	Click this button to close the selected tab of the Run tool window and terminate the current process.
	Up/down the Stack Trace Ctrl+Alt+Up Ctrl+Alt+Down	Click this button to navigate up or down in the stack trace and have the cursor jump to the corresponding location in the source code.
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location in the source code.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.
	Execute Groovy Code (Ctrl+Enter)	Run code, entered in the console.
	Help F1	Use this icon or shortcut to open the corresponding help page.

Play Framework (Play Console)

Tools | Play with Playframework

Note that the Play console is available for Play 1.x framework in the IntelliJ IDEA version 11.0. For later versions of IntelliJ IDEA refer to Play 2.x framework and its features.

The Tools | Play with Playframework command and the Play Framework tab or view in the Run tool window are available only under the following conditions:

- The Playframework Support plugin is enabled. See Enabling and Disabling Plugins .
- A Play application is currently open in IntelliJ IDEA. (In technical terms, the directory `<play_dir>\framework\lib` and the file `<play_dir>\framework\play-<version>.jar` are included in the dependencies of the module that represents your Play application.)

Use the Play Framework tab or view for running commands of the `play` command-line utility (the Play console) and performing other, related tasks.

The console workspace is divided into two parts. In the upper part, the console output is shown. The lower part is for the input; this is where you type the `play` commands (omitting the leading `play`). The parts are separated with a horizontal line.

- Toolbar
- Context menu for the output area
- Context menu for the input area

Toolbar

Icon	Shortcut	Description
	Ctrl+F2	Use this icon or shortcut to stop the Play application which is currently running (the one previously started by means of the `play run` command).
	Ctrl+F4	Use this icon or shortcut to close the Play Framework tab or view. If the Play application is currently running, you can also select to stop it in the dialog that opens.
	Enter	Use this icon or shortcut to run the current `play` command.
	F1	Use this icon or shortcut to open the corresponding help topic.
	Ctrl+Alt+Up	
	Ctrl+Alt+Down	
		Use this icon to turn the soft wrapping mode in the console on or off.
		Use this icon to scroll down to the end of the console output.
		Use this icon to send the console output to the default printer.
		Use this icon to clear all the contents of the output area.

Context menu for the output area

Command	Shortcut	Description
Copy	Ctrl+C	Use this command or shortcut to copy the selected text fragment to the clipboard. Note that this command is available only if something is currently selected in the output area. If nothing is selected, you can still use Ctrl+C to copy all the contents of the output area to the clipboard.
Copy Reference	Ctrl+Shift+Alt+C	Use this command or shortcut to place the text `\Play Framework:1` onto the clipboard.
Compare with Clipboard		Use this command to compare selected text or all the contents of the output area (if nothing is currently selected) with the contents of the clipboard.
Search with Google		Use this command to search the Internet for the selected text fragment using Google. Note that this command is available only if something is currently selected in the output area.
Fold Lines Like This		Use this command to create a new folding pattern based on the current line. See also, Console .
Clear All		Use this command to clear all the contents of the output area.

Context menu for the input area

Command	Shortcut	Description
Cut	Ctrl+X	Use this command or shortcut to remove the selected text fragment from the input area and to place it onto the clipboard. Note that this command is available only if something is currently selected in the input area. If nothing is selected, you can still use Ctrl+X to remove the whole command (without the leading `play`) and to place it onto the clipboard.
Copy	Ctrl+C	Use this command or shortcut to copy the text fragment selected in the input area to the clipboard. Note that this command is available only if something is currently selected in the input area. If nothing is selected, you can still use Ctrl+C to copy the whole command (without the leading `play`) to the clipboard.
Copy Reference	Ctrl+Shift+Alt+C	Use this command or shortcut to place the text `\Play Framework:1` onto the clipboard.
Paste	Ctrl+V	Use this command or shortcut to paste the last of the clipboard entries into the input area.
Paste from History	Ctrl+Shift+V	Use this command or shortcut to open the Choose Content to Paste dialog. This dialog lets you select the clipboard entry to be pasted into the input area.
Paste Simple	Ctrl+Shift+Alt+V	Use this command or shortcut to paste the last of the clipboard entries into the input area as plain text.

Compare with Clipboard	Use this command to compare selected text with the contents of the clipboard. If nothing is currently selected, the whole `play` command (without the leading `play`) is used for comparison.
Search with Google	Use this command to search the Internet for the selected text fragment using Google. Note that this command is available only if something is currently selected in the input area.
Fold Lines Like This	Use this command to create a new folding pattern based on the current line. See also, Console .

Seam Tool Window

View | Tool Windows | Seam

Note This window is available only for modules with a Seam facet .

The Seam tool window provides a structured view of your Seam resources. These include annotated Java classes, the appropriate xml files, and the libraries referenced in your Seam components.

The resources are grouped by modules. The modules are the top hierarchical elements in the tree view.

You can expand and collapse the nodes in the tree view, access the dependency diagrams for your Seam components, and open the elements shown in the tool window in the editor.

- Toolbar
- Context menu
- Opening elements in the editor

Toolbar

Use the toolbar buttons to expand and collapse the nodes in the tree view.

ItemTooltip	Shortcut	Description		
			Expand All	`Ctrl+NumPad Plus`	Click this button to expand all the nodes.
			Collapse All	`Ctrl+NumPad -`	Click this button to collapse all the nodes.

Context menu

Command	Description
Show Seam Components Dependencies	Use this command to open the diagram that shows the Seam components and dependencies between them for a selected module.

Opening elements in the editor

You can open the elements shown in the Seam tool window in the editor. To do that, select the element of interest and press `F4`.

To open the elements which are the "leaves" of the tree (i. e. the ones at the bottom of the hierarchy), you can also use a double click.

Spring Tool Window

View | Tool Windows | Spring

The Spring tool window lets you look at your project from the Spring perspective. It consists of two tabs:

- Beans tab
- MVC tab

Each tab consists of a set of panes where the contents of the selected item is shown in the pane to the right. The documentation (and, if applicable, a diagram) for the selected item is shown in the rightmost pane.

The toolbars in each tab control the amount and the type of data to be displayed.

The context menu commands allow you to switch between the panes and open items in the editor.

You can filter the information in the panes by entering a search string.

In this topic:

- Beans tab
  - Beans tab toolbar
- MVC tab
  - MVC tab toolbar
- Context menu commands
- Filtering information
- Beans icons

Beans tab

The Beans tab allows you to view definitions for the Spring beans used in your project, and see how they are related to other beans.

The leftmost pane shows a list of modules that your project consists of. When you select a module, the list of filesets (contexts) appears on the right. The next pane shows the list of configuration files included in the selected context. The next pane shows the list of Spring beans defined in the selected configuration file. The rightmost pane shows documentation for the selected bean, and a diagram that shows this bean's relation to other beans.

Beans tab toolbar

Icon	Tooltip	Description
	Show modules	Select this option if you want the panes showing the modules, contexts and configuration files to be displayed. If this option is disabled, only the beans pane showing all beans used in your project is displayed.
	Show filesets	Select this option if you want the panes showing the filesets (contexts) and configuration files to be displayed. If this option is disabled, only the modules pane and the beans pane are displayed.
	Show configuration files	Select this option if you want the panes showing the configuration files to be displayed. If this option is disabled, only the modules pane, the filesets pane and the beans pane are displayed.
	Show Implicit Beans	Select this option if you want implicit beans to be displayed in the beans pane. Implicit beans are service beans added by the Spring framework and are not defined explicitly by the user.
	Show Infrastructure Beans	Select this option if you want infrastructure beans to be displayed in the beans pane. Infrastructure beans are service beans that form the structure of a context by pointing to configuration files where other beans are defined.
	Show Bean Documentation	Select this option if you want documentation for the selected bean to be displayed in the rightmost pane. The documentation pane shows the type of the selected bean (Spring Bean, specific namespace element, @Component, @Repository, etc.), the bean class, some bean attributes (prototype, @Qualifier, etc.), and the path to the file where this bean is declared.
	Show Bean Graph	Select this option if you want a diagram showing how the selected bean is related to other beans to be displayed in the rightmost pane.

MVC tab

The MVC tab allows you to view controller mappings for the Spring MVC framework.

The leftmost pane shows a list of modules that your project consists of. When you select a module, the list of controllers that belong to this module appears on the right. The next pane shows the mappings defined for the selected controller. The rightmost pane shows documentation for the selected mapping.

MVC tab toolbar

Icon	Tooltip	Description
	Show modules	Select this option if you want the panes showing the modules to be displayed. If this option is disabled, only the controllers pane and the mappings pane are displayed.
	Show controllers	Select this option if you want the controllers pane to be displayed. If this option is disabled, only the modules pane and the mappings pane are displayed.
	Request Method	Click this icon to filter mappings by the HTTP method. Only the mappings that implement the methods selected in the popup menu are displayed in the mappings pane.
	Show documentation	Select this option if you want documentation for the selected mapping to be displayed in the rightmost pane. The documentation pane shows the mapping between the URL and the file that must be opened for this URL, and the HTTP method that this mapping implements.

Context menu commands

The context menu is available in all panes of the Spring Tool Window and the Spring Tool Window except for the documentation pane.

Command	Shortcut	Description
Previous	Left	Use this command to switch to the pane on the left of the current one.
Next	Right	Use this command to switch to the pane on the right of the current one.
Edit	Enter	Use this command to view or edit the cource code for the selected item in the editor. If a module is selected, this command opens this module's settings in the Project Structure dialog.

Filtering information

You can filter data in any pane of the Beans tab and the MVC tab except for the documentation pane. Place the cursor in a pane and start typing. As a result, only the items whose names contain the specified string will be displayed.

Beans icons

The icons before the name of each bean in the list indicate the bean type:

- : beans defined in XML files.
- : Spring auto-discoverable beans declared through `@Component` annotations.
- : service beans added by the Spring framework and not defined explicitly by the user.

Spy-js Tool Window

The tool window is available only when the Spy-js plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The tool window opens when you launch a run configuration of the type Spy-js or Spy-js for NodeJS . As soon as the Spy-js tool captures an event, it shows the event itself, its details, and stack-traces. By clicking an event, you can open the trace file, which is a prettified source file with the script that the event launched.

The tool window consists of a common toolbar and two tabs with panes:

- Trace Run Tab
- Trace Proxy Server Tab

Toolbar

Item	Tooltip and shortcut	Description
	Rerun Ctrl+F5	Click this button to stop the current trace session and run it again. When the session is canceled externally by clicking , the button toggles to . Note, that after the session restarts, you have to refresh the traced page in the browser to start capturing events.
	Rerun F9	This button substitutes for if the previous trace session was canceled externally by clicking . Click to initiate a new session. Note, that after the session restarts, you have to refresh the traced page in the browser to start capturing events.
	Stop Ctrl+F2	Click this button to terminate the current process externally by means of the standard `shutdown` script.
	Restore Layout	Click this button to to have the changes to the current layout abandoned and return to the default state.
	Pin	Use to pin or unpin the tab. If a tab is pinned, the results for the next command are shown on a new tab.
	Close Ctrl+Shift+F4	Click this button to close the selected tab of the Run tool window and terminate the current process.
	Help F1	Use this icon or shortcut to open the corresponding help page.

Trace Run Tab

The tab consists of a toolbar and three panes: Events Pane, Event Stack Pane, and Quick Evaluation Pane.

On this page:

- Events Toolbar
- Events Pane
  - Context Menu of a Document Node
  - Context Menu of an Event or Script
  - Configuring the Range of Events to Capture
    - Defining a new event filter
    - Activating a filter
    - Adding an event to an exclusion filter on the fly
- Event Stack Pane
  - Context Menu of a Script or Function
- Synchronization between the Panes and the Editor
- Quick Evaluation Pane
  - Context Menu of Function Call Details

Events Toolbar

Use the buttons on the toolbar to control the range of events to capture, configure their presentation, and navigate through the list of captured events.

Item	Tooltip and shortcut	Description
	Expand all `Ctrl+NumPad Plus`	Click this button to have all the nodes in the list expanded.
	Collapse all `Ctrl+NumPad -`	Click this button to have all the nodes in the list collapsed.
	Up the Stack Trace	Click this button to navigate to the previous traced page in the stack trace.
	Down the Stack Trace	Click this button to navigate to the next traced page in the stack trace.
	Autoscroll to source	Press this toggle button to have the list in the Events pane automatically synchronized with the Editor. – When the button is pressed: as soon as you click an event in the Events pane, the details of the event are displayed in the Event Stack pane and the script that is invoked by the event is opened in the editor automatically. When you navigate through the Event Stack with the Autoscroll to Trace mode turned on, the corresponding files are also automatically opened in the editor with the calling functions highlighted. – When the button is released: the script that is invoked by the event is opened in the editor only upon double-clicking the event in the Event Stack pane.
	Capture Events	Click this button to configure the range of events to be captured and shown in the Events list. By default, the Spy-js tool captures all events on all opened Web pages, excluding https secure web sites, unless you have specified a URL address explicitly in the run configuration. The Events pane of the Spy-js tool window shows all captured events. If for some reasons you do not want to have all events captured, you can suppress capturing some of them by applying user-defined event filters. When you click , the pop-up list shows all the available filters, the currently applied filter is marked with a tick. By default the Capture All predefined filter is applied. To stop capturing events without stopping the application, choose Mute All. The application is still running but the Events pane shows the last captured event. This is helpful if you want to analyze a script and therefore need it to be displayed in the Events pane instead of being removed as new events are captured. To define a custom event filter: 1. Click , and then choose Edit Capture Exclusions from the list. 2. In the Spy-Js Capture Exclusions Dialog that opens, click the Add toolbar button in the left-hand pane. 3. In the right-hand pane, specify the filter name in the Exclusion name field and configure a list of exclusion rules. To add a rule, click , the Add Condition to Exclusion dialog box opens. Type a pattern in the Value/pattern text box, in the Condition type drop-down list specify whether the pattern should be applied to event types or script names. Note, that glob pattern matching is used. When you click OK, IntelliJ IDEA brings you to the Spy-Js Capture Exclusions Dialog . To edit a rule, select it in the list, click , and update the rule in the dialog box that opens. To remove a rule, select it in the list and click . To activate a filter, set a tick next to the required filter in the list.
		Click this button to remove all or some events from tracing and have the corresponding trace files closed in the editor. Choose one of the following options on the drop-down list that is displayed: – Remove all: choose this option to cancel tracing of all captured events without closing the trace files in the editor. – Close all trace files: choose this option to have all trace files in the editor closed but keep tracing the corresponding event. To remove an event or script from tracing and close the corresponding trace files in the editor, choose Remove on the context menu of the event or script. – Remove all inactive: choose this option to remove all nodes for pages that are not active anymore (for example, because the pages have been closed in browser). – Save trace: choose this option to save an image of the current session. The `.json` files that store the calls and properties of the session are compressed into a `zip` archive. Upon request, when you choose Load trace, the `.json` files are extracted from the archive and loaded into Spy-js. – Load trace: choose this option to have a previously saved image of a tracing session loaded into Spy-js. The `.json` files that store the calls and properties of that session are extracted and imported. Note that a loaded image does not restore the session because no scripts are actually executed. All you can do is analyze the flow and properties of previously executed code. – Close all trace files on session stop: choose this option to have all trace files in the editor closed when you click the Stop button to stop the tracing session externally by means of the standard `shutdown` script.

- Enable Spy-js autocomplete and magnifier: choose this option to have the basic completion list expanded with runtime data (Spy-js autocompletion) and to get the possibility to evaluate expressions without actually running a debugging session (Spy-js magnification). By default, the functionality is turned off.
  The term Spy-js autocompletion denotes expanding the basic completion list with suggestions retrieved from the runtime data. The Spy-js autocompletion functionality is available from source files for the code that has already been executed (highlighted green in the corresponding trace file).

  When you position the caret at a symbol in the source file and press `Ctrl+Space`, Spy-js retrieves data from the browser or from the running NodeJS application and merges it with the basic completion list according to the following rules:

  1. If an object both is present on the basic completion list and is retrieved from the runtime, the variant that provides more information about parameters, attributes, their type, etc. remains on the list.
  2. Objects retrieved by Spy.js are shown on top of the list and marked with the icon. If a retrieved object is specific for a browser, the object is marked with the icon and with the icon of this browser.

  The term Spy-js magnification denotes evaluating expressions without actually running a debugging session. When you click the expression in question or position the caret at it and press `Ctrl+Alt+F8`, a tooltip is displayed below the expression showing the expression value. If Spy-js retrieves several values, click + icon in the tooltip to expand the list of values.

  The magnification functionality is available from source files for both executed and not yet executed code.
- Spy-js supports source maps, which means that you can now jump from the Event Stack pane right to the original source code in ECMAScript 6, TypeScript or CoffeeScript and observe what code fragments were executed. Use the following options to configure the way source maps are treated:
  - Enable source map look-up: choose this option to enable navigation to the ECMAScript 6, TypeScript or CoffeeScript source code using the source maps generated during transpilation.
  - Enable source map generation: choose this option to generate source maps for everything to map the instrumented code. Choose this option if you are going to debug the original code in Chrome Dev Tools or FireFox FireBug development tools.
  - Always open source mapped trace if available: choose this option to have Spy-js try to open the mapped trace file when you invoke navigation from an event to its caller.

Events Pane

The pane shows a tree of captured events. The top-level nodes represent documents that are Web pages involved in tracing. When you hover the mouse over a document, IntelliJ IDEA displays a tooltip with the URL address of the document, the browser in which it is opened, and the operating system the browser is running on. The document node is also supplied with an icon that indicates the browser in which it is opened.

Under each document node, events detected on the page and scripts started from it are listed. Events of the same type are grouped into visual containers. The header of a container displays the name of the events grouped in it, the average execution time across all the events within the container, and the number of events inside the container. You can expand each node and inspect the individual events in it.

Script file names have different colour indicators to help distinguishing between them when working with the Event Stack pane. By hovering your mouse over a script file name, you can see the full script URL.

Once an event is clicked, its call stack is displayed in the Event Stack pane. The stack is represented by a tree of function calls.

Context Menu of a Document Node

Menu item	Description
Remove	Choose this option to cancel tracing all the scripts on the selected page and remove the selected node with all the items under it from the Events pane. All the currently opened trace files remain opened in the editor.
Remove all children	Choose this option to delete the items under the selected page but keep tracing it so that new events from the page are still received. The document node itself remains in the Event pane, and all the currently opened trace files remain opened in the editor.
Remove and close trace file(s)	Choose this option to cancel tracing all the scripts on the selected page, remove the selected node and all the items under it from the Events pane, and close the corresponding trace files in the editor.
Close trace file(s)	Choose this option to close all the currently opened trace files that correspond to the selected document node and items under it. The document node and the items under it remain in the Events pane.
Refresh the page in browser	Choose this option to reload the page that corresponds to the selected document node. Tracing of the selected node is abandoned, a new document node for tracing the same page is created, and the old node becomes inactive.
Try closing the page in browser	Choose this option to close the page that corresponds to the selected node. Tracing of the selected node is abandoned, and the node becomes inactive.
Show application dependency diagram	Choose this option to build a diagram with the dependencies within the entire application. – The diagram is opened in a separate editor tab. The nodes in the diagram represent your project files, while the edges represent the fact that there's one or more functions in the source file that invoke functions in the target file. – To examine the details of a node or an edge, select the node or the edge in question and view its Details tree in a dedicated pane in the upper right-hand corner of the editor. The pane displays the connecting function combinations, along with event(s) the calls are made within and the number of calls made.

Context Menu of an Event or Script

Menu item	Description
Mute event	Choose this option to add an event to an exclusion filter on the fly.
Mute script	Choose this option to add a script to an exclusion filter on the fly.
Remove	Choose this option to cancel tracing the selected event or script, remove the selected item from the Events pane, but leave the corresponding trace files opened in the editor.
Add label	Choose this option to set a timestamp label. Timestamp labels help you to analyze your code execution within a specific period of time. For example, you can set two timestamp labels and view which events were captured between them. Or on the contrary, you can locate the events that were not captured within a certain period of time although you expected them to be and thus detect performance problems.
Show event dependency diagram	Choose this option to build a diagram with the dependencies in which the event selected event is involed. – The diagram is opened in a separate editor tab. The nodes in the diagram represent your project files, while the edges represent the fact that there's one or more functions in the source file that invoke functions in the target file. – To examine the details of a node or an edge, select the node or the edge in question and view its Details tree in a dedicated pane in the upper right-hand corner of

the editor. The pane displays the connecting function combinations, along with event(s) the calls are made within and the number of calls made.

Configuring the Range of Events to Capture

By default, the Spy-js tool captures all events on all opened Web pages, excluding https secure web sites, unless you have specified a URL address explicitly in the run configuration. The Events pane of the Spy-js tool window shows all captured events. If for some reasons you do not want to have all events captured, you can suppress capturing some of them by applying user-defined event filters. All the available filters are listed upon clicking the Capture Events button on the toolbar, the currently applied filter is marked with a tick. By default the Capture All predefined filter is applied.

To stop capturing events without stopping the application, choose Mute All. The application is still running but the Events pane shows the last captured event. This is helpful if you want to analyze a script and therefore need it to be displayed in the Events pane instead of being removed as new events are captured.

You can define new custom filters or add event patterns to existing filters on the fly.

Defining a new event filter

1. Click the Capture Events button on the toolbar, and then choose Edit Capture Exclusions from the list.
2. In the Spy-Js Capture Exclusions Dialog that opens, click the Add toolbar button in the left-hand pane.
3. In the right-hand pane, specify the filter name in the Exclusion name field and configure a list of exclusion rules.
   To add a rule, click , the Add Condition to Exclusion dialog box opens. Type a pattern in the Value/pattern text box, in the Condition type drop-down list specify whether the pattern should be applied to event types or script names. Note, that glob pattern matching is used. When you click OK, IntelliJ IDEA brings you to the Spy-Js Capture Exclusions Dialog .

   To edit a rule, select it in the list, click , and update the rule in the dialog box that opens. To remove a rule, select it in the list and click .

Activating a filter

– Click and set a tick next to the required filter in the list. If no filters are configured or none of the available filters fits the task, create a new filter as described above.

Adding an event to an exclusion filter on the fly

While navigating through the tree of already captured events in the Events pane, you may come across some events or scripts that you definitely do not want to trace. You can create a filter as described above but in this case you will have to leave the pane. With IntelliJ IDEA, you can create an exclusion rule based on any event or script, as soon as you have detected such event or script, right from the Events pane. The rule will be either added to the currently applied filter or a new filter will be created if the current setting is Capture All.

– To add an event to an exclusion filter on the fly, select the event to exclude and choose Mute <event name> event or Mute <script name> file.
  If a user-defined filter is currently applied, the new rule is added to it silently. If Capture All is currently active, the Spy-Js Capture Exclusions Dialog opens, where you can create a new filter based on the selected event or script or choose an existing filter and add the new rule to it.

Event Stack Pane

Once an event in the Events pane is clicked, its call stack is displayed in the Event Stack pane. The stack is represented by a tree of function calls. Each tree node represents the invoked function. Node text contains the total execution time, the script file name and the function name. When you click a node, the Quick Evaluation pane shows additional function call details, parameter values and return value, occurred exception details if there was one during the function execution.

The pane is synchronized with the editor, so you can navigate from an item in the stack tree to the corresponding trace file or source file.

– A trace file is a write-protected prettified version of the script selected in the Events pane or the script whose function is double clicked in the Event Stack pane. A trace file is named `<file name>.js.trace`. When you double click an item in the stack tree or select it and choose Jump to Trace on the context menu of the selection, the corresponding trace file opens in the editor with the cursor positioned at the clicked function. Another approach is to press the Autoscroll to Trace toggle button and select various stack nodes. In this case, the trace file opens when you click an event or script in the Events pane.
  You can not only jump to a function but also to the place in the code where it was called from. To do that, select the required item and choose Jump to Caller on the context menu.

  The contents of the file are highlighted to display the code execution path of the selected stack node.

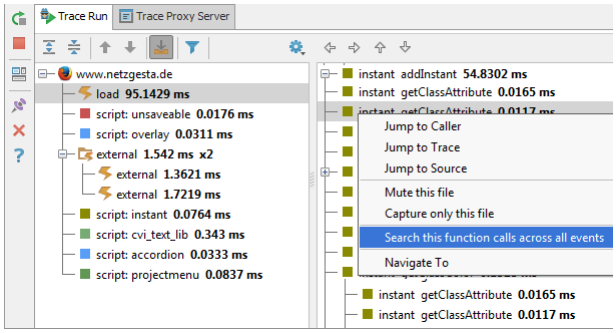
– When you are tracing an application with ECMASript6, CoffeeScript, and TypeScript code, Spy-js also generates mapped trace files. These are EcmaScript 6, TypeScript, or CoffeeScript trace files with the extensions `.ts.trace`, `.coffee.trace`, or `.js.trace`. The fragments of code in these files are highlighted as if they were really executed.

– You can also navigate to the source file displayed as is, without prettifying, by selecting an item in the Event Stack pane and choosing Jump to Source on the context menu of the selection. If the traced site is mapped with a IntelliJ IDEA project, IntelliJ IDEA detects the corresponding local file according to the mapping and opens this file in the editor. If you are tracing a site that is not mapped to a IntelliJ IDEA project, IntelliJ IDEA opens the read-only page source, just as if you chose View Page Source in the browser.
  When the traced site is mapped with a IntelliJ IDEA project, IntelliJ IDEA opens the source file on any attempt to edit the opened trace file.

Context Menu of a Script or Function

Item	Description
Jump to Caller	Choose this option to navigate to the fragment in the trace file from where the currently selected item was called. When you are tracing an application with ECMASript6, CoffeeScript, and TypeScript code, IntelliJ IDEA opens either the trace JavaScript file or the mapped trace file (TypeScript, CoffeeScript, or ECMAScript6): – If the Always open source mapped trace if available option is selected, the corresponding mapped trace file opens. – If the Always open source mapped trace if available option is not selected, the JavaScript trace file opens.
Jump to Trace	Choose this option to navigate to the definition of the currently selected item in the trace file.
Jump to	Choose this option to navigate to the definition of the currently selected item in the source file.

Source	
Mute this File	Choose this option to add the selected script to an exclusion filter on the fly, see Configuring the Range of Events to Capture .
Capture only this file	
Search this function calls across all events	Choose this option to navigate between the calls of a function within the whole trace (across all the traced events). This means that if you are tracing 5 pages in the browser and the Events pane, accordingly, shows 5 document nodes, IntelliJ IDEA searches for the calls of the selected function under all these nodes and displays the number of found calls of the function in the Status bar. The number of found calls is displayed in the Status bar, and the toolbar shows four previously hidden navigation chevron buttons.



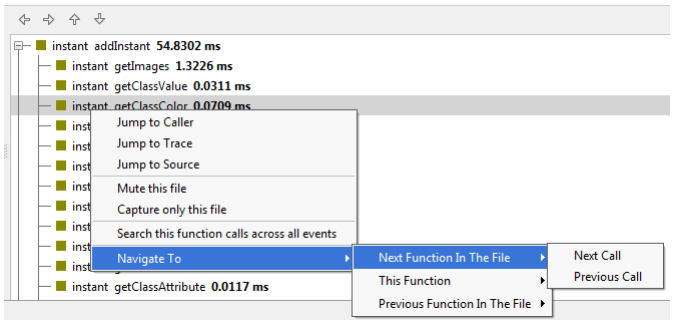
Use the chevron buttons to navigate within the found calls:

- To jump to the first detected call, click .
- To jump to the last detected call, click .
- To jump to the next detected call, click . The Status bar shows a message: `Occurrence <number> of <total number of detected calls>`
- To jump to the previous detected call, click .

The search results are reset and the search toolbar is hidden when you invoke another advanced search or navigation.

Also keep in mind that the number of call occurrences is calculated when you choose the Search this function calls across all events option. As you analyze the detected calls, the time passes, new events are captured, and the first detected call can happen to be already removed from the stack which means that it is no longer available for navigation.

Navigate to	Use the options under this item to move through the whole stack based on calls and locate the functions that have not been called, that is, locate the fragments of code that have not been executed and analyze the reason for them to be skipped. The following six actions are available: move to the next/previous call of the next/current/previous function in a trace file. The full list of actions is available from the context menu in the Event Stack pane. Moving to the next and previous calls of the selected function, to the previous call of the previous function, and to the next call of the next function are also available from the navigation toolbar of the Event Stack pane.



When you choose one of these actions, the cursor jumps to the call in the stack. If the Autoscroll to Trace toggle button is pressed, the corresponding trace file opens automatically with the cursor positioned at the call.

Synchronization between the Panes and the Editor

The Events and Event Stack panes are synchronized: when you click an event or script in the Events pane, its call stack is displayed in the Event Stack pane. To have also the corresponding trace file opened in the editor, press the Autoscroll to Trace toggle button on the toolbar.

The Event Stack pane is synchronized with the editor: when you click an item in the stack tree twice, the corresponding trace file opens in the editor with the cursor positioned at the clicked function.

To synchronize the Events pane directly with the editor, press the Autoscroll to Trace toggle button on the toolbar. In this case, as soon as you click a node in the Events pane, its call stack is displayed in the Event Stack pane and the corresponding trace file is opened in the editor. With the Autoscroll to Trace mode turned on, when you navigate through the Event Stack the corresponding files are also automatically opened in the editor with the corresponding functions highlighted.

Quick Evaluation Pane

When you click a node in the Event Stack pane, the Quick Evaluation pane shows additional function call details, parameter values and return value, occurred exception details if there was one during the function execution.

Context Menu of Function Call Details

The context menu is available from all items displayed in the pane.

Item	Description
Inspect	Choose this option to open the Inspect dialog box.
Copy Value	Choose this option to copy the value of the selected node to the clipboard.
Compare Value with Clipboard	Choose this option to open the Differences viewer for files which displays the value of the selected node and the value in the clipboard so you can compare them.
Copy Name	Choose this option to copy the name of the selected node to the clipboard.

Spy-js Capture Exclusions Dialog

The dialog box opens when you click the Capture Events button toolbar button in the Events pane of the Spy-Js Tool Window . In this dialog box, define custom filters to configure the range of events displayed and traced in the tool window. See Tracing with Spy-Js for details.

The dialog box consists of two panes. The left-hand, Exclusions, pane shows a list of already existing user-defined event filters. The right-hand, Exclusion, pane shows the details of the filter selected in the Exclusions pane.

Item	Description
Exclusions pane	The pane shows a list of all currently available user-defined filters. – To define a new event filter, click on the toolbar, then specify the filter name and exclusion rules in the Exclusion pane. – To create a filter based on an already existing one, make a copy of the source filter by selecting it and clicking , and then rename and edit the copy as required in the Exclusion pane. – To temporarily disable a filter, clear the check box next to it. – To remove a filter from the list, select it and click .
Exclusion pane	On this pane, configure custom event filters. For each filter, specify its name and create a list of exclusion rules. – To add a rule, click , the Add Condition to Exclusion dialog box opens. Type a pattern in the Value/pattern text box, in the Condition type drop-down list specify whether the pattern should be applied to event types or script names. Note, that glob pattern matching is used. When you click OK, IntelliJ IDEA brings you to the Spy-Js Capture Exclusions Dialog . – To edit a rule, select it in the list, click , and update the rule in the dialog box that opens. – To remove a rule, select it in the list and click .

Trace Proxy Server Tab

The tab consists of a toolbar and the console area that shows system information and error messages, information about the status of the Spy-js session, the availability of the proxy server, the currently active frames ("workers"), etc.

Trace Proxy Server Toolbar

Item	Tooltip and shortcut	Description
	Stop Trace Proxy Server	Click this button to terminate the currently running process externally by means of the standard `shutdown` script.
	Up/Down the Stack Trace Ctrl+Alt+Up Ctrl+Alt+Down	Use these buttons to navigate through the stack trace from one reported error to another. These buttons are available only when errors are reported, for example, if the configuration file is corrupted.
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location in the source code.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.

Structure Tool Window, File Structure Popup

Structure tool window

This tool window displays the structure of a file currently opened in the editor and having the focus, or selected in the Project tool window.

For diagrams, this tool window shows the diagram preview.

View | Tool Windows | Structure

`Alt+7`

File Structure pop-up window

This pop-up window displays the structure of a file, currently opened in the editor and having the focus.

Navigate | File Structure

`Ctrl+F12`

Both views help quickly navigate through the files' structure. Refer to the section Navigating with Structure Views .

This section describes the buttons on the title bar of the tool window and the options on the context menu of the title bar. Turn these options on and off to have elements of certain types hidden or shown and configure the way they are presented.

The buttons on the title bar are common for all language contexts. The set of options on the context menu depends on the context.

- Title Bar
- Java
- HTML, XML
- JavaScript, CoffeeScript, ActionScript
- Properties
- PHP

Tip Refer to the section Symbols for the member icons in the tree view of a file in the Structure tool window.

Title Bar

The buttons on the title bar are common for all language contexts.

Item	Tooltip	Description
	Collapse All `Ctrl+NumPad -`	Click this button to have all the nodes in the tool window collapsed.
	Expand All `Ctrl+NumPad Plus`	Click this button to have all the nodes in the tool window expanded.
		Click this button to open the context menu and configure the appearance of the tool window, its viewing mode , and the way it presents the structure of the current file by turning the menu items on or off.
	Hide `Shift+Escape`	Click this button to hide the tool window.

Java

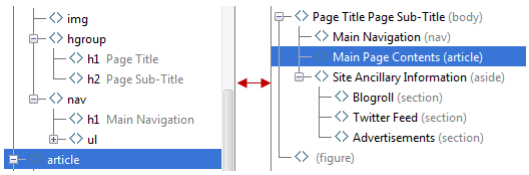
Icon	Tooltip	Description
	Sort by Visibility	Click this button to have the items sorted by their visibility in the following order: public - protected - package local - private.
	Sort Alphabetically	Click this button to have the elements within a class sorted alphabetically.

Tip When both (Sort by Visibility) and (Sort Alphabetically) buttons are pressed, the items in the view are grouped according to their visibility levels. Within each visibility level group, the items are sorted alphabetically. When both buttons are released, the items are shown in the order they appear in the code.

	Group Methods by Defining Type	Click this button to have all the methods that override/implement the methods of a particular class/interface grouped under the node that corresponds to this class/interface.
	Show Properties	Click this button to show getters, setters, and fields in the tree view.
	Show Inherited	Click this button to display all the methods and fields inherited by the current class and accessible from it. The inherited members are displayed gray to distinguish them from the members defined in the current class.
	Show Anonymous	Click this button to have the inner anonymous classes shown in the tree view.
	Show Fields	Click this button to have all fields (properties) shown in the tree.
	Show non-public	Click this button to have all non-public class members displayed. Turn off the option to hide all non-public members.
	Collapse All `Ctrl+NumPad -`	Click this button to have all the nodes in the tool window collapsed.
	Expand All	Click this button to have all the nodes in the tool window expanded.

	Ctrl+NumPad Plus	
	Autoscroll to Source	Click this button to enable automatic navigation to the line of source code that corresponds to the selected node when the focus switches to the editor.
	Autoscroll from Source	Click this button to have IntelliJ IDEA automatically move the focus in the Structure tool window to the node that corresponds to the code where the cursor is currently positioned in the editor.

HTML, XML

Icon	Tooltip	Description
	Sort Alphabetically	Click this button to have the elements within a class sorted alphabetically.
	HTML5 Outline	Click this button on to view HTML 5 outline of a HTML file: 
	Collapse All Ctrl+NumPad -	Click this button to have all the nodes in the tool window collapsed.
	Expand All Ctrl+NumPad Plus	Click this button to have all the nodes in the tool window expanded.
	Autoscroll to Source	Click this button to enable automatic navigation to the line of source code that corresponds to the selected node when the focus switches to the editor.
	Autoscroll from Source	Click this button to have IntelliJ IDEA automatically move the focus in the Structure tool window to the node that corresponds to the code where the cursor is currently positioned in the editor.

JavaScript, CoffeeScript, ActionScript

Icon	Tooltip	Description
	Sort Alphabetically	Click this button to have the elements within a class sorted alphabetically.
	Group Methods by Defining Type	Click this button to have all the methods that override/implement the methods of a particular class/interface grouped under the node that corresponds to this class/interface.
	Show Fields	Click this button to have all fields (properties) shown in the tree.
	Show Inherited	Click this button to display all the methods and fields inherited by the current class and accessible from it. The inherited members are displayed gray to tell them from the members defined in the current class.
	Collapse All Ctrl+NumPad -	Click this button to have all the nodes in the tool window collapsed.
	Expand All Ctrl+NumPad Plus	Click this button to have all the nodes in the tool window expanded.
	Autoscroll to Source	Click this button to enable automatic navigation to the line of source code that corresponds to the selected node when the focus switches to the editor.
	Autoscroll from Source	Click this button to have IntelliJ IDEA automatically move the focus in the Structure tool window to the node that corresponds to the code where the cursor is currently positioned in the editor.

Properties

Icon	Tooltip	Description
	Sort Alphabetically	Click this button to have the elements within a property file sorted alphabetically.
	Sort by Type	Click this button to have the elements within a class sorted according to their types.
	Group by prefix	Click this button to have the elements within a property file grouped according to their prefixes.
	Collapse All Ctrl+NumPad -	Click this button to have all the nodes in the tool window collapsed.
	Expand All Ctrl+NumPad Plus	Click this button to have all the nodes in the tool window expanded.
	Autoscroll to Source	Click this button to enable automatic navigation to the line of source code that corresponds to the selected node when the focus switches to the editor.
	Autoscroll from Source	Click this button to have IntelliJ IDEA automatically move the focus in the Structure tool window to the node that corresponds to the code where the cursor is currently positioned in the editor.

PHP

Title Bar

Besides the buttons that are common for all language contexts, the title bar in the PHP context contains two additional toggle buttons.

Icon	Tooltip	Description
PHP		Toggle this button to view the hierarchy of PHP elements. The button is always present on the toolbar but is enabled only when the current PHP class or file contains fragments of HTML code.

HTML		Toggle this button to view the hierarchy of HTML elements in a PHP class or file. The button is only available is the current PHP class or file contains fragments of HTML code.

Context Menu

Icon	Tooltip	Description
	Sort by Visibility	Click this button to have the items sorted by their visibility in the following order: public - protected - package local - private.
	Sort Alphabetically	Click this button to have the elements within a class sorted alphabetically.
	Show Inherited	Click this button to display all the methods and fields inherited by the current class and accessible from it. The inherited members are displayed gray to distinguish them from the members defined in the current class.
	Show Includes	Click this button to have all files included through `include` or `require` statements shown in the tree.
	Show Fields	Click this button to have all fields (properties) shown in the tree.
	Show Constants	Click this button to have constants shown in the tree.
	Collapse All `Ctrl+NumPad -`	Click this button to have all the nodes in the tool window collapsed.
	Expand All `Ctrl+NumPad Plus`	Click this button to have all the nodes in the tool window expanded.
	Autoscroll to Source	Click this button to enable automatic navigation to the line of source code that corresponds to the selected node when the focus switches to the editor.
	Autoscroll from Source	Click this button to have IntelliJ IDEA automatically move the focus in the Structure tool window to the node that corresponds to the code where the cursor is currently positioned in the editor.

Struts Assistant Tool Window

View | Tool Windows | Struts Assistant

The Struts Assistant tool window appears when you open a project that contains a module with enabled Struts support . Use this tool window to manage elements , tiles , and validators in a Struts application.

The Struts Assistant tool window has three tabs:

- The Struts tab for managing Struts elements in the `struts-config.xml` configuration file.
- The Tiles tab for managing Struts tiles in the `tiles-defs.xml` definition file.
- The Validator tab for defining action-specific validation rules.

The Struts Assistant tool window displays three views:

- The Structure Tree shows the structure of the Struts configuration file `struts-config.xml` depending on the current tab.
- The Properties Table shows the attributes of the node selected in the Structure Tree.
- The Struts Web Flow Diagram graphically presents the relations among the elements of your application.
  The Struts Web Flow Diagram appears in the Struts tab only.

The views are synchronized. If a node is selected in the Structure Tree, the Properties Table shows the node's attributes. If a node has a corresponding diagram object, this diagram object is selected as well.

Common Toolbar Buttons

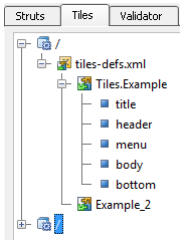
Item	Tooltip and shortcut	Description
	Autoscroll to Source	Click this button to enable navigation from the selected element in the Structure Tree to the corresponding source code block in the `struts-config.xml` configuration file (opening it, if necessary).
	Autoscroll from Source	Click this button to enable navigation from a source code block in the `struts-config.xml` configuration file to the corresponding node in the Structure Tree.
	Expand all Ctrl+NumPad Plus	Use these buttons to have all nodes expanded or collapsed.
	Collapse all Ctrl+NumPad –	

Tiles Tab

View | Tool Windows | Struts Assistant

The tab is available from the Struts Assistant tool window .

Use this tab to manage the Struts tiles in your application.



Root node tiles-defs.xml

Item	Name and tooltip	Description
	tiles-defs.xml	The root node of the application tiles' structure tree.
Context menu (same for all nodes)		
F4	Jump to source	Select this menu item to switch to the code block in the `tiles-defs.xml` file, text view, that corresponds to the selected item in the tree.
	Expand All	Click this button to expand all nodes.
	Collapse All	Click this button to collapse all nodes.

Tile definition

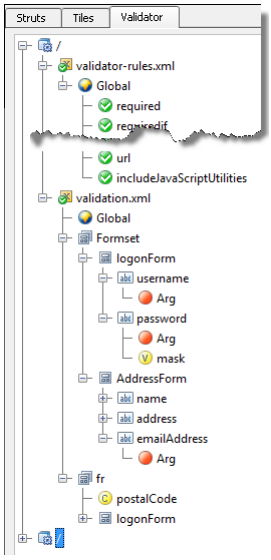
Item	Name and tooltip	Description
	Tile	The node under which an application tile is defined.
Context menu		
	Add Put	Select this menu item to add a new `<put>` element to the tile definition.
	Add Put List	Select this menu item to add a list of `<put>` elements to the tile definition.
	Remove Definition	Select this menu item to remove the tile definition.

Validator Tab

View | Tool Windows | Struts Assistant

The tab is is available from the Struts Assistant tool window .

Use this tab to define validation procedures for specific actions.



validator-rules.xml node

Item	Name and tooltip	Description
	validator-rules.xml	The node under which the definitions of standard validation routines are placed.
	Context menu (same for all nodes)	
F4	Jump to source	Select this menu item to switch to the code block in the `validator-rules.xml` file, text view, that corresponds to the selected item in the tree.
	Expand All	Click this button to expand all nodes.
	Collapse All	Click this button to collapse all nodes.
	Global	The node under which the definitions of global validation routines are placed.
	Context menu	
	Add Constant	Select this menu item to add a constant to the selected validation routine.
	Add Validator	Select this menu item to add a standard routine validator.
	Remove Global/Validator	Select this menu item to remove the entire Global node or the selected Validator.
	validation.xml node	

Struts Tab

View | Tool Windows | Struts Assistant

The tab is available from the Struts Assistant tool window .

Use this tab to manage Struts elements in the `struts-config.xml` configuration file.

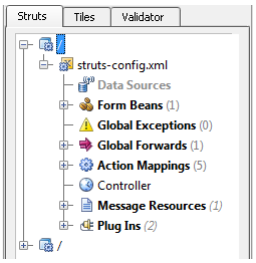
The tab displays three views:

- Structure Tree
- Properties Table
- Web Flow Diagram

The views are synchronized. If a node is selected in the Structure Tree, the Properties Table shows the node's attributes. If a node has a corresponding diagram object (for example, an Action cell or a Forward edge), this diagram object is selected as well.

Structure Tree

The view shows the Struts elements that are used in your application as nodes of a structure tree. As you create, edit, and remove these nodes and their subordinate items, these changes are immediately reflected in the `struts-config.xml` configuration file.

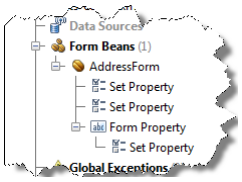


Item	Name and tooltip	Description
	struts-config.xml	The root node of the application structure tree.
Context menu (the same for all nodes)		
F4	Jump to source	Select this menu item to switch to the code block in the `struts-config.xml` file, text view, that corresponds to the selected item in the tree.
	Expand All	Click this button to expand all the nodes in the tree.
	Collapse All	Click this button to collapse all the nodes in the tree.

The view shows the following Struts element nodes:

- Form Beans Node
- Global Exceptions Node
- Global Forwards Node
- Action Mappings Node
- Controller Node
- Message Resources Node
- Plugins Node

Form Beans Node

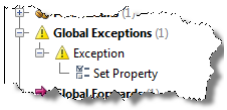


Item	Name and tooltip	Description
	Form Beans	Under this node, manage form beans and their properties. The following actions are available through the context menu of the Form Beans node: • Add Form Bean — Select this menu item to add a new form bean to the application. • Remove Form Beans — Select this menu item to remove all the form beans from the application.
	Form Bean	A form bean node. The following actions are available through the context menu of a Form Bean:

	Add Form Property	Select this menu item to add a form property to the bean.
	Add Set Property	Select this menu item to add a set property to the bean or to a form property.
	Remove Bean/ Form Property/ Set Property	Select this menu item to remove the entire bean or its form or set property.

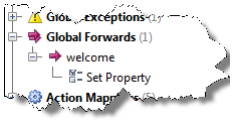
	Form Property	Bean's form property.
	Set Property	A set property of the bean or a form property.

Global Exceptions Node



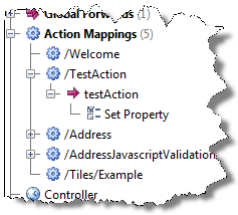
Item	Name and tooltip	Description
	Global Exceptions	Under this node, manage global exceptions and their properties. The following actions are available on the context menu of the Global Exceptions node: • Add Global Exception: Select this menu item to add a new global exception to the application. • Remove Global Exceptions: Select this menu item to remove all the global exceptions from the application.
	Global Exception	A global exception node. The following actions are available on the context menu of a Global Exception: • Add Set Property: Select this menu item to add a set property to the global exception. • Remove Exception/ Set Property: Select this menu item to remove the entire global exception or its set property.
	Set Property	A set property of the global exception.

Global Forwards Node



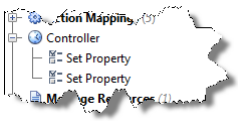
Item	Name and tooltip	Description
	Global Forwards	Under this node, manage global forwards and their properties. The following actions are available on the context menu of the Global Forwards node: • Add Global Forward: Select this menu item to add a new global forward to the application. • Remove Global Forwards: Select this menu item to remove all the global forwards from the application.
	Global Forward	A global forward node. The following actions are available on the context menu of a Global Forward: • Add Set Property: Select this menu item to add a set property to the global forward. • Remove Exception/Set Property: Select this menu item to remove the entire global forward or its set property.
	Set Property	A set property of the global forward.

Action Mappings Node



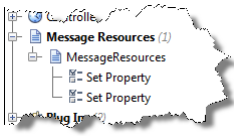
Item	Name and tooltip	Description
	Action Mappings	Under this node, manage the action mappings within the application, mapping properties, forwards, and exceptions. The following operations are available on the context menu of the Action Mappings node: • Add Action — Select this menu item to dd a new action to the application. • Remove Action Mappings — Select this menu item to remove all the action mappings from the application.
	Action	An action node. The following operations are available on the context menu of an Action: • Add Exception — Select this menu item to add an exception to the action. • Add Forward — Select this menu item to add a forward to the action. • Add Set Property — Select this menu item to add a set property to the action or actions's forward or action's exception.
	Exception	An exception defined within the action.
	Global Forward	A forward within the action.
	Set Property	A set property of the action, or action's forward, or action's exception.

Controller Node



Item	Name and tooltip	Description
	Controller	Under this node, manage the properties of the application controller. The following operations are available on the context menu of the Controller node: • Add Set Property — Select this menu item to add a set property to the controller. • Set Property — A set property of the controller. • Remove Controller/ Set Property — Select this menu item to remove the controller or its set property from the application.

Message Resources Node

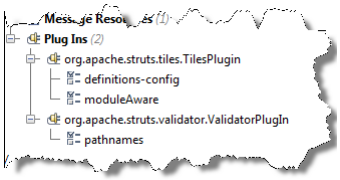


Item	Name and tooltip	Description
	Message Resources	Under this node, manage the message resources of the application. The following operations are available on the context menu of the Message Resources node: • Add Message Resource — Select this menu item to add a message resource to the application. • Remove Message Resources — Select this menu item to remove the all message resources from the application.
	Message Resource	A message resource of the application. The following operations are available on the context menu of a Message Resource:

Add Set Property	Select this menu item to add a set property to the message resource.
Remove Message Resources/ Set Property	Select this menu item to remove the message resource or its set property from the application.

	Set Property	A set property of the message resource.

Plugins Node



Item	Name and tooltip	Description
	Plugins	Under this node, manage the application plugins. The following operations are available on the context menu of the Plugins node:

Add Plugin	Select this menu item to add a new plugin to the application.
Remove Plugins	Select this menu item to remove all plugins from the application.

	Plugin	A plugin node. The following operations are available on the context menu of a Plugin:

Add Set Property	Select this menu item to add a set property to the plugin.
Remove Plugin/Set Property	Select this menu item to remove the entire plugin or its set property.

	Set Property	A set property of the plugin.

Properties Table

The view shows the attributes of the node selected in the Structure Tree.

Item	Description
Name	The names of the attributes of the element selected in the Structure Tree.
Value	The values of the attributes of the element selected in the Structure Tree.

Web Flow Diagram

The view graphically presents the relations among the elements of your application. The diagram features common graph view elements.

Forwards and exceptions are displayed as directed graph edges.

Icons used on the Web Flow Diagram

Item	Description
	Struts Action
	Struts JSP page
	Struts global forward
	Struts global exception

Choose Class

The dialog opens in various cases when you need to select a class that implements a specific element.

The dialog contains two tabs:

– Search by Name

– Project

Search by Name Tab

Use the tab to search a relevant class by specifying its name or part of the name.

Item	Description
Search pattern area	In this text field, type a part of the name of the relevant class.
Search results area	The area shows a list of the classes that meet the search pattern. The contents of the area change dynamically as you type.

Project Tab

Use this tab to select the relevant class in the project tree.

Time Tracking Tool Window

View | Tool Windows | Time Tracking

Item	Tooltip and Shortcut	Description
+	Open Task `Shift+Alt+N`	Create new task. Refer to the section Opening and Creating Tasks .
−	Remove Task	Delete selected tasks from the list.
⌄	Show closed tasks	If this button is pressed, the list of tasks includes closed tasks. This button works as the second pressing keys `Shift+Alt+N`.
	Auto mode	If this button is pressed, time tracking automatically turns on for the current active task, and the button is disabled. If this button is not pressed, you have to start timer for the active task yourself.
/	Start/stop timer for the active task	Click this button to start or stop timer for the active task. When the timer is started, the button toggles to (Stop timer for active task). This button is disabled, when time tracking works in automatic mode (the button is pressed.)
	Post work item to bug tracker	Click this button send duration of work and optional comment to the bug tracker.
	Show time spent from last post of work item	

Thumbnails Tool Window

Project tool window | context menu | Show Thumbnails

`Ctrl+Shift+T`

The Thumbnails tool window provides the functions similar to those of an image browser. It shows thumbnails for folders and image files, and lets you perform related navigation and image management tasks.

To open the tool window, use the Show Thumbnails command in the Project tool window (`Ctrl+Shift+T`). Then, unless you close the tool window, you can show and hide it as described in the section Manipulating the Tool Windows .

– Title bar context menu
– Title bar icons
– Toolbar icons
– Content pane: context menu commands

Title bar context menu

The title bar context menu provides the options for controlling the tool window viewing modes . It also contains the commands for associating the tool window with a different tool window bar , resizing and hiding the tool window.

To access the menu, right-click the window name (Thumbnails).

Note that most of the menu options may alternatively be accessed by means of the title bar icons .

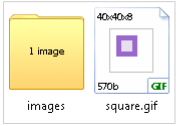
Item	Shortcut	Description
Pinned, Docked, Floating, Split Mode		These options let you control general appearance and behavior of the tool window, see Viewing Modes .
Move to		To associate the tool window with a different tool window bar , select this command, and then select the destination tool window bar (Top, Left, Bottom or Right).
Resize		To resize the tool window by moving one of its borders, select this command, and then select the necessary Stretch to option. Note that this command is not available in the floating mode.
Hide	`Shift+Escape`	Use this command to hide the tool window.

Title bar icons

Item	Shortcut	Description
		Use this icon to open the menu for changing the tool window viewing modes .
	`Shift+Escape`	Use this icon or shortcut to hide the tool window . When used in combination with the `Alt` key, clicking this icon hides all the tool windows attached to the same tool window bar .

Toolbar icons

The toolbar icons provide access to the most frequently used functions available in the tool window. The same and other functions are available as context menu commands in the content pane.

Item	Shortcut	Description
	`Backspace`	Use this icon or shortcut to move one level up in the folder hierarchy. The path to the current folder is shown on the title bar to the right of the tool window name.
	`Ctrl+numpad*`	Use this icon or shortcut to turn the Recursive option on or off. If this option is off, subfolders of the current folder and image files located in the root of the current folder are shown. If this option is on, the subfolders are not shown; the image files located in the current folder and all its subfolders are shown. The following picture shows the contents of the same folder with the Recursive off and on. Recursive: OFF  Recursive: ON
		Use this icon to show or hide a "chessboard". The "chessboard" (a checkered area) is shown underneath the images so that you can see transparent image areas for `.gif` and `.png` files. The following picture shows the same thumbnail with the "chessboard" shown and hidden.

Ctrl+F4 Use this icon to close the tool window.

Content pane: context menu commands

When you right-click an item in the content pane, the context menu for this item is shown.

The following table lists and briefly explains the main context menu commands for thumbnails. Other commands, functionally, are similar to those in the Project tool window .

Item	Shortcut	Description
Browse	Enter	For a folder: use this command to view the folder contents (subfolders and images). The same command may alternatively be accessed by double-clicking a folder.
Jump to Source	F4 or Enter	For an image file: use this command to view the selected file in the editor. The same command may alternatively be accessed by double-clicking an image thumbnail.
Level up	Backspace	Use this command to move one level up in the folder hierarchy.
Recursive	Ctrl+numpad*	Use this command to turn the Recursive option on or off.
Show or Hide Chessboard		Use this command to show or hide the chessboard .
Close thumbnails	Ctrl+F4	Use this command to close the tool window.
Jump to External Editor	Ctrl+Alt+F4	For an image file: use this command to open the image in an external editor.

TODO Tool Window

View | Tool Windows | TODO

`Alt+6`

IntelliJ IDEA scans your project for comments in the source code that match the TODO patterns defined in the TODO dialog and displays results in the TODO tool window.

The TODO tool window consists of following tabs:

- Project tab that show the TODO items for the whole project.
- Current File tab.
- Scope Based tab that enables viewing TODO items pertaining to a certain scope, selected from the drop-down list, and ignoring the other items.
- Current Changelist, if version control is enabled.

This tool window helps you view, sort and group the TODO items in a convenient way, navigate to the source code and keep fixes under the version control.

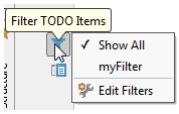
On this page:

- Toolbar buttons
- Context menu commands
- Title bar context menu and buttons

Toolbar buttons

Item	Tooltip and shortcut	Description
	Previous TODO `Ctrl+Alt+Up`	Navigate to the previous TODO item.
	Next TODO `Ctrl+Alt+Down`	Navigate to the next TODO item.
	Help `F1`	Use this icon or shortcut to open the corresponding help page.

	Expand all
	`Ctrl+NumPad Plus`
	Collapse all
	`Ctrl+NumPad -`
		Use these buttons to have all nodes expanded or collapsed.

	Autoscroll to Source	Toggle the Autoscroll to source mode. When this button is pressed, every time the node is focused, the corresponding line of source code is highlighted in the editor.
	Filter TODO items	Click this button to select the desired filter from the list, or invoke the TODO dialog and edit the list of TODO patterns and filters as required. 
	Preview Usages	If this button is pressed, a pane to the right shows the source code of the selected file, with the corresponding TODO item highlighted.

Tip The following buttons are available in the Project and Scope Based tabs.

	`Ctrl+D`	If this button is pressed, the TODO items show under the corresponding module or library node.
	`Ctrl+P`	If this button is pressed, the TODO items show under the corresponding packages.
	`Ctrl+F`	If this button is pressed, the TODO items show as a flat list. Thus, if a package is somewhere deep within your project, you do not need to dig deep into the hierarchy.

Context menu commands

Item	Keyboard Shortcut	Description
Jump to Source	`F4`	Navigate to the selected usage in the source code.
Local History		Show Local History submenu for the selected search result. Refer to the Local Version Control procedures for details.
<VCS>		Show menu of the VCS, associated with the directory. See Version Control Procedures and Reference for details.

Title bar context menu and buttons

The title bar context menu provides the options for controlling the tool window viewing modes . It also contains the commands for associating the tool window with a different

tool window bar , resizing and hiding the tool window.

To access the menu, right-click the window title bar.

Note that most of the menu options may alternatively be accessed by means of the title bar buttons.

Toolbar icon	Context menu command	Description
	Pinned, Docked, Floating, Split Mode	These options let you control general appearance and behavior of the tool window, see Viewing Modes .
	Move to	To associate the tool window with a different tool window bar , select this command, and then select the destination tool window bar (Top, Left, Bottom or Right).
	Resize	To resize the tool window by moving one of its borders, select this command, and then select the necessary Stretch to option. Note that this command is not available in the floating mode.
 Shift+Escape	Hide	Use this command to hide the tool window. Tip When used in combination with the Alt key, clicking this button hides all the tool windows attached to the same tool window bar .

TypeScript Compiler Tool Window

On this page:

- Current Errors Pane
- Project Errors Pane
- Toolbar
- Context Menu
- Console

Current Errors Pane

The pane shows a list of all the discrepancies detected in the file which is opened in the active editor tab. At the top the full path to the file is displayed. For each discrepancy, IntelliJ IDEA provides a brief description and information about the line number where the error occurred. In addition to this you can navigate to the code in question right from the corresponding error message by choosing Jump to Source on the context menu.

Project Errors Pane

The pane shows a list of all the discrepancies detected in all the files in the current project after you run the built-in compiler against the entire project by clicking on the toolbar. The error messages are grouped by files in which they were detected. For each discrepancy, IntelliJ IDEA provides a brief description and information about the line number where the error occurred. In addition to this you can navigate to the code in question right from the corresponding error message by choosing Jump to Source on the context menu.

Toolbar

The toolbar is common for the Current Errors and the Project Errors panes.

Item	Tooltip and Shortcut	Description
	Compile Current File	Click this button to run the compiler against the entire file opened in the active editor tab. This approach is helpful if you do not want the compiler to wake up on any change as you edit your code and for this purpose the Track Changes check box on the TypeScript page of the Settings dialog box is cleared. If you click this button from the Project Errors pane, after compilation IntelliJ IDEA switches to the Current Errors pane.
	Compile All	Click this button to run the compiler against all the TypeScript files in the current project. If you click this button from the Current Errors pane, after compilation IntelliJ IDEA switches to the Project Errors pane.
	Close Ctrl+Shift+F4	Click this button to terminate the compiler and close the tool window.
	Expand all Ctrl+NumPad Plus	Use these buttons to have all nodes expanded or collapsed.
	Collapse all Ctrl+NumPad -	
	Clear All	Click this button to remove all the error messages from the currently active pane.

Context Menu

The context menu is common for the Current Errors and the Project Errors panes.

Item	Description
Jump to source	Choose this option to open the file where the selected error occurred during compilation and navigate to the fragment of code which caused the error.
Copy	Choose this option to copy the selected error message with the information on the file, the line, and the column where the error occurred during compilation.

Console

The tab shows the log of the built-in compiler since the tool window was opened.

Version Control Tool Window

View | Tool Windows | Version Control

This tool window is available if version control is enabled for your project.

The tool window accomodates several viewes that display VCS-realted information and allow you to manage changelists, perform VCS-specific actions, view changes made by other team members, etc.

The tool window consists of several tabs:

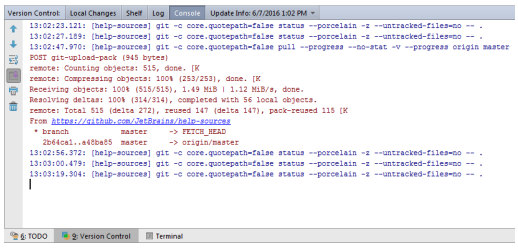
- Console tab: this tab shows the results of executing VCS-related commands.
- Local Changes tab: this tab is always present and shows the list of files that have been modified locally and have not been committed to the repository yet.
- History tab: this tab is added to the Version Control tool window when the Show History command is invoked through VCS | <specific_VCS>.
- Integrate to Branch Info View tab: this view is available after running integration with the Run status after update setting specified.
- Log tab: this tab is only available if you are using Git or Mercurial as your version control system. It shows all changes committed to all branches of the local and remote repositories, or to a specific branch or repository.
- Repository and Incoming tabs: the Repository tab shows the changes committed to the repository under the VCS roots within the current project. The Incoming tab shows the changes commited to the repository by other team members, and not checked out locally.
- Shelf tab: this tab is added to the tool window when you shelve a change or a changelist.
- Update Info tab: this tab becomes available when local information is synchronized with the server.

Console Tab

View | Tool Windows | Version Control | Console

The tab displays:

- Version control-related commands generated based on the settings you specify through the IntelliJ IDEA interface.
- The results of executing version control-related commands.



Toolbar

Item	Tooltip and Shortcut	Description

	Up the stack trace / Down the stack trace
	`Ctrl+Alt+Up`
	`Ctrl+Alt+Down`
		Click these buttons to navigate up or down in the stack trace and have the cursor jump to the corresponding location.

	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.

Context Menu Options

Item	Description
Compare with Clipboard	Opens the Clipboard vs Editor dialog box that allows you to view the differences between the selection from the editor and the current clipboard content. This dialog is a regular comparing tool that enables you to copy the line at caret to the clipboard, find text, navigate between differences and manage white spaces.
Fold Lines Like This	Opens the Console dialog that allows you defining the lines to be folded to hide extraneous information.
Copy URL	Choose this command to copy the current URL to the system clipboard. This command only shows on a URL, if it is included in an application's output.
Create Gist	Choose this command to open the Create Gist dialog box.
Clear All	Clears the output window.

Local Changes Tab

VCS | Show Changes View - Local Changes

View | Tool Windows | Version Control - Local Changes

`Alt+9`

The Local Changes tab shows the files that have been modified locally and have not yet been committed to the repository.

Note You can assign a custom shortcut for the Show Local Changes action in Settings | Keymap | Version Control Systems to open the Local Changes tab.

Use this tab to commit and revert changes, manage changelists , view differences , view changes in UML Class diagram , and clean up locked folders.

In this topic:

- Toolbar
- Changelists pane
  - Context menu of a selection
- Preview Diff Pane

Toolbar

Item	Tooltip and Shortcut	Description
	Refresh `Ctrl+F5`	Click this button to refresh the status of all files in your workspace, both made through IntelliJ IDEA or through any other application. Note that if you are using Perforce as your version control system, only the status of files modified through IntelliJ IDEA will be updated. This approach improves performance, as it does not require connecting to the server, but it does not let you get an update on the changes made outside IntelliJ IDEA, for example through the p4v client application. If you want to get an update on all changes to your workspace, use the Force Refresh option.
	Force Refresh	This button is only available if you are using Perforce as your version control system. Click this button to refresh the status of files in your workspace, both modified through IntelliJ IDEA or through other applications.
	Commit Changes	Click this button to check in the selected change or changelist. You can also attach and detach Perforce jobs to/from changelists via the Commit Changes dialog.
	Revert `Ctrl+Alt+Z`	Click this button to roll back the selected changes.
	New Changelist `Alt+Insert`	Click this button to create a new changelist .
	Delete Changelist `Delete`	Click this button to delete the selected changelist. Note that you cannot delete the default changelist.
	Set Active Changelist	Click this button to make the selected changelist active . The active changelist is highlighted.
	Move to Another Changelist `F6`	Click this button to move the selected file to another changelist.
	Show Diff `Ctrl+D`	Click this button to view the differences between your local version of the selected file and its latest version in the repository.
	Show Changes `Ctrl+Shift+Alt+D`	Use this button to show classes from the selected changelist in a UML Class diagram (for details see Viewing Changes as Diagram).

	`Ctrl+NumPad Plus`
	
	`Ctrl+NumPad -`
		Click these buttons to expand or collapse all nodes.

	Group by Directory `Ctrl+P`	Click this button to display the changed files grouped by directories. If the button is released, the changed files are grouped by changelists.
	Copy `Ctrl+C`	Click this button to copy the path to the selected file to the clipboard.
	Show Ignored Files	Click this button to show the Ignored files node with the list of existing files ignored by the VCS.
	Configure Ignored Files	Click this button to configure the list of files that will be ignored by your version control system.
	Preview Diff	Click this button to have IntelliJ IDEA open or close the Preview Diff pane to compare the current file with the latest committed revision.
	`F1`	Click this button to show the corresponding IntelliJ IDEA help page.

Changelists pane

This pane shows all your changelists, and the files that have been modified in each changelist.

If new files have been added to your project that have not yet been added to a version control system, the Unversioned Files node appears under which all such files are listed. If you have a large number of unversioned files (over 50), they are not displayed in the changelists pane. Instead, the Click to browse link appears. Click this link to open the Unversioned Files dialog to see the list of files that have not been added to a version control system. You can quickly delete unversioned files from the Changelists pane or the Unversioned Files dialog by pressing `Delete`, or add them to the VCS by pressing `Ctrl+Alt+A`.

Context menu of a selection

Item	Shortcut	Description
Commit Changes	N/A	Select this option to check in the selected file or changelist. You can also attach and detach Perforce jobs to changelists via the Commit Changes dialog.
Revert	`Ctrl+Alt+Z`	Select this option to roll back the selected changes.
Move to Another Changelist	`F6`	Select this option to move the selected item to another changelist.
Show Diff	`Ctrl+D`	Select this option to view the differences between your local copy and the latest version in the repository.
Jump to Source	`F4`	Select this option to open the selected file(s) in the editor.
New Changelist	`Alt+Insert`	This option is only available if a changelist is selected, not single files. Select this option to create a new changelist .
Delete	N/A	This option is only available if single files are selected, not a changelist.
Check Out	N/A	This option is only available if a file under the Modified without Checkout node is selected. Use this option to check out the selected file from the repository.
Add to VCS	N/A	This option is only available if a file under the Unversioned Files node is selected. Use this option to add the selected files to your version control system.
Ignore	N/A	This option is only available if a file under the Unversioned Files node is selected. Use this option to ignore the selected file if you want to leave it unversioned.
Create Patch	N/A	Select this option to create a patch .
Shelve Changes	N/A	Select this option to shelve the selected changes .
Shelve in Perforce	N/A	This option is only available if you are using Perforce as a version control system. Select this option to shelve your changes in Perforce. You will be asked to select which files you want to shelve and provide a description. After you've shelved your changes, the corresponding changelist will appear. You can unshelve it any time from the changelist's context menu.
Refresh	`Ctrl+F5`	Select this option to refresh the status of files in your workspace.
Local History	N/A	Select this option and choose one of the following from the popup menu: – Show History: select this option to View local history of the selected file or folder. – Put Label: select this option to add a lable to the current version of the selected file or folder.
<Specific version control system>	N/A	Select this option to invoke a popup menu with options specific for the version control system you are using.

Preview Diff Pane

This pane opens when you click the Preview Diff button on the toolbar. In this pane you can examine the changes made to the selected file compared to its base revision.

The pane consists of two areas:

– The affected code as it was in the base revision.

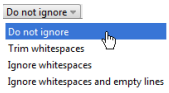
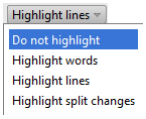
– The affected code as it is after a change has been made.

To close the pane, click the Preview Diff button once again.

Item	Tooltip and Shortcut	Description
	Previous Difference / Next Difference `Shift+F7` `F7`	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, IntelliJ IDEA suggests to click the arrow buttons `F7` / `Shift+F7` once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings . This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.

	Compare Previous/Next File
	
	`Alt+Left`
	`Alt+Right`
		Click these buttons to compare the local copy of the previous/next file with its update from the server.
		Tip These buttons are available only when the Differences Viewer is invoked from the Version Control tool window.

	Jump to Source `F4`	Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.
Viewer type		Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to – Edit, for example, type text. Note that one can change text only in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line. – Perform the Apply/Append/Revert actions.

Whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text. – Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default. – Trim whitespaces: `("\t", " ")` , if they appear in the end and in the beginning of a line. – If two lines differ in trailing whitespaces only, these lines are considered equal. – If two lines are different, such trailing whitespaces are not highlighted in the By word mode. – Ignore whitespaces: white spaces are not important, regardless of their location in the source code. – Ignore whitespaces and empty lines: the following entities are ignored: – all whitespaces (as in the 'Ignore whitespaces' option) – all added or removed lines consisting of whitespaces only – all changes consisting of splitting or joining lines without changes to non-whitespace parts. For example, changing `a b c` to `a \n b c` is not highlighted in this mode.
Highlighting mode		Select the way differences granularity is highlighted. The available options are: – Highlight words: the modified words are highlighted – Highlight lines: the modified lines are highlighted – Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one. – Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	Collapse unchanged fragments	Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.
	Synchronize scrolling	Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.
	Editor settings	Click this button to reveal the drop-down list of available check commands. Select or clear these check commands to show or hide line numbers, indentation guides, white spaces, and soft wraps. Note that same commands are duplicated in the context menu of the differences viewer gutter. Right-click the gutter to show the menu of check commands.
	Show diff in external tool	Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff Tools settings page.
	Help F1	Click this button to show the corresponding help page.

The most useful shortcuts in the Diff Pane are the following:

Shortcut	Description
Ctrl+Shift+D	Use this keyboard shortcut to show the popup menu of the most commonly user diff commands.
Ctrl+Tab	Use this keyboard shortcut to switch between the left and the right panes.
Ctrl+Shift+Tab	Use this keyboard shortcut to select the position obtained by Ctrl+Tab in the opposite pane.
Ctrl+Z / Ctrl+Shift+Z	Use this keyboard shortcut to undo/redo a merge operation. Conflicts will be kept in sync with the text.

Log Tab

VCS | Show Changes View - Log

View | Tool Windows | Version Control - Log

`Alt+9`

The tab is only available if your project is integrated with Git or Mercurial .

This tab shows all changes committed to all branches of the local and remote repositories, or to a specific branch or repository.

Note that the log is updated automatically in the background mode on every change (a commit, fetch, rebase, etc.), even if the Log tab is closed.

Note You can assign a custom shortcut for the Show VCS Log action in Settings | Keymap | Version Control Systems to open the Log tab.

The tab contains the following panes:

- The Commits pane is located on the left of the tool window and shows the commits to all or selected branches from the local and remote repositories.
- The Changed Files pane is located on the right of the tool window and shows the list of files that were modified and committed within the selected commit.

Commits Pane

The commits pane consists of three areas:

- Commits
- Commit Details
- Toolbar . Some of the toolbar commands are duplicated in the context menu .

Commits

This area shows a list of all commits performed to the selected branch, or to all branches. For each commit, the list shows the commit message, the author, and the commit date. The latest commit in each branch is supplied with a color label with the name of the branch in which it was performed.

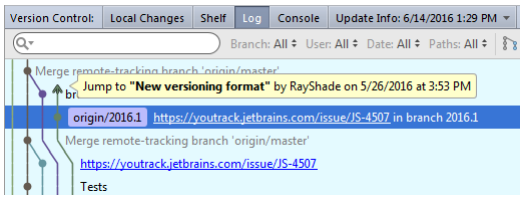
There are the following labels:

- local (Mercurial) and regular (Git/Mercurial) for tags.
- tip (Mercurial) for the latest revision in the repository.
- HEAD (Git/Mercurial) for the current working revision.

For Git, the color of the label depends on the branch type (local or remote).

For Mercurial, there are different color labels for bookmarks, open heavy branches and closed heavy branches.

Note that clicking an arrow takes you to the next commit in a long branch:



In multi-root projects, the colored stripe on the left indicates which root the selected commit belongs to (each root is marked with its own color). Hover the mouse cursor over the colored stripe to invoke a tip that shows the root path. You can also enable the Show Root Names option if you want to expand the Roots column with full root names.

Committed changelists often correspond to issues in tracking systems. You can jump to such issues in a browser right from the Commits pane. This functionality is available if:

- The pattern of the bug tracking system is specified in the Issue Navigation Settings dialog box.
- The corresponding issue number is mentioned in the commit message.

After issue navigation has been configured, issue numbers in commit messages are rendered as links. Clicking such link brings you to the corresponding page of your issue tracker.

Commit Details

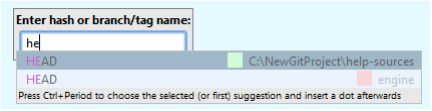
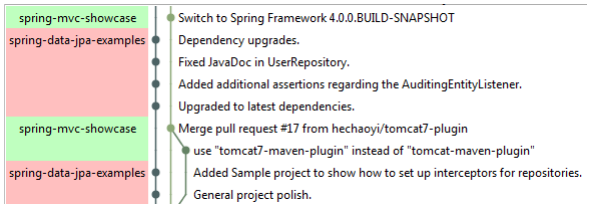
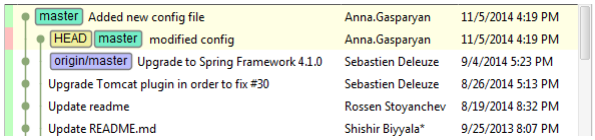
This area is displayed when the Show Details option is enabled.

This area shows the details on the commit selected in the Commits list, such as date, time, author, root and branches.

If the selected commit is contained in more than six branches, only the first six are displayed and the Show All link appears that you can click to expand a complete list of branches.

Toolbar

Item	Tooltip and Shortcut	Description
Filter	N/A	Use this text box to search through the list of commits. You can enter full or parts of commit names or messages, revision numbers, or regular expressions. As you type, the commits containing your search string are displayed. To finalize the search, press `Enter`. You can quickly switch the focus to the search field by pressing `Ctrl+L`.

	N/A	Click this button to show previously used search patterns.
	N/A	Click this button to clear the search history list.
Branch	N/A	Use this drop-down list to filter the commits by the branch. If you want to see commits from all local and remote branches, select All.
User	N/A	Use this drop-down list to filter the commits by the author. To view the commits of a specific author, click Select and start typing the author's name. To view commits by all users, select All.
Date	N/A	Use this drop-down list to filter the commits by a time-frame or a specific date. To view the commits made on a specific date, click Select and specify the date. To view commits made on all dates, select All.
Paths	N/A	Use this drop-down list to filter the commits by the folder (for projects that have one root), or by the root and folder (for multi-root projects). To view the commits to a specific folder, click Select Folders and specify the folder name. For multi-root projects, you can also select the check-box next to one or several roots in the Roots section.
	IntelliSort	If this option is enabled, you get a more convenient way to view merges by displaying the incoming commits first, directly below the merge commit.
	Collapse linear branches	Click this button to collapse all branches on the graph so that a dotted line is shown instead of successive commits. It is also possible to collapse an individual expanded branch by clicking it.
	Expand all branches	Click this button to expand all collapsed branches to show successive commits on the graph. It is also possible to expand an individual collapsed branch by clicking it.
	Show long edges	If this option is enabled, long branches are displayed in full, even if there are no commits in them. If this option is disabled (by default), long branches are replaced with a down arrow.
	Refresh Ctrl+F5	Click this button to refresh the information in the Commits pane.
	Show details	Click this button to show the Commit details pane .
	Go to Hash/Branch/Tag Ctrl+F	Click this button and specify a hash, tag or branch you want to jump to in the text field.  You can select a reference with the same name from different repositories. The name of each repository is displayed on the right side of the list along with its color indicator.
	Quick settings	Click this button to invoke one of the following commands: – Show Branches Panel: enable this option if you want a panel listing all branches to be displayed: You can click on a label in the branches panel to jump to the corresponding branch in the Commits list. – Show Root Names: enable this option if you want to expand the Roots column on the left showing full root names. Note that this option is only available for multi-root projects.  – Highlight My Commits: enable this option if you want your commits to be highlighted with the yellow background in the commits list. 
	Cherry-pick (for Git) Graft (for Mercurial)	Click this button to apply the changes from the selected commit to the current branch .
	Highlight non-picked commits	Note that this command is only available if you are using Git as your version control system. Click this button to highlight the commits from the selected branch that have not yet been applied to the current branch.

Context menu commands

Item	Description	Available in
Copy Revision Number	Use this command to copy the revision number of the selected commit to the clipboard.	Git Mercurial
Create Patch	Use this command to create a patch based on the selected commit.	Git Mercurial
Cherry-pick (Git)	Use this command to apply the changes from the selected commit to the current	Git

	branch .	
Graft (Mercurial)		Mercurial
Checkout Revision	Use this command to check out the state of files recorded in the selected commit.	Git
Update to Revision	Use this command to change your working copy parent revision to the selected commit. New commits will carry on from the revision (commit) you update to.	Mercurial
New Branch	Use this command to create a new branch based on the selected commit.	Git Mercurial
New Tag	Use this command to add a new tag to the selected commit.	Git Mercurial
Branch <branch_name> / Branches	This command appears for all branches that point to the selected commit (Branch <branch_name> if there is one branch, or Branches if there are multiple branches) and provides the same options as the ones available from the Branches popup and submenu : – Checkout as new local branch – Compare – Rebase onto – Merge – Delete If the Control repositories synchronously option is enabled, and the selected branch exists in multiple repositories, an additional menu option named In All Repositories appears that allows you to perform the same operations in all repositories simultaneously.	
Reset Current Branch to Here	Use the command to reset the current branch head to the selected commit. In the Git Reset dialog that opens, select the mode in which the working tree will be updated.	Git
Open in GitHub	Use this command to open the page that corresponds to the selected commit on GitHub .	Git
MQ	Use this submenu to manage Mercurial Queues : – Import: use this command to turn the selected changeset into a patch. – Goto patch: use this command to open the MQ: <project_name> tab that shows a queue of patches that have not been applied yet. – Rename Patch: use this command to rename the selected patch. – Finish Patches: use this command to turn the selected patch into a permanent changeset.	Mercurial

Changed files pane

This pane shows a list of files that were modified within the currently selected commit.

Toolbar

Many of the options available in the toolbar are duplicated in the context menu.

Item	Tooltip and Shortcut	Description
	Show Diff Ctrl+D	Click this button to open the Differences viewer for files where you can compare the current and the previous revision of the selected file.
	Show Diff with Local	Click this button to show the differences between the selected revision of the selected file and its current local copy.
	Edit Source F4	Click this button to open the local copy of the selected file for editing.
	Open Repository Version	Click this button to open the repository version of the selected file for editing.
	Revert Selected Changes	Click this button to roll back the changes in the selected file.
	Group by Directory Ctrl+P	Click this button to transform a flat list of files into a tree of packages with files.
	Expand All/Collapse All Ctrl+NumPad Plus	Click this button to expand/collapse all nodes. Note that these buttons are only available only when tree-view is enabled.

Context menu commands

Item	Tooltip and Shortcut	Description
	Show Diff Ctrl+D	Click this button to open the Differences viewer for files where you can compare the current and the previous revision of the selected file.
	Show Diff with Local	Click this button to show the differences between the selected revision of the selected file and its current local copy.
	Edit Source F4	Click this button to open the local copy of the selected file for editing.
	Open Repository	Click this button to open the repository version of the selected file for editing.

	Version	
	Revert Selected Changes	Click this button to roll back the changes in the selected file.

History Tab

The History tab is added to the Version Control tool window on invoking the Show History command for a file or directory through the menu of a particular VCS. A new tab is created for each file or directory, with the name File <name> History. The set of toolbar buttons differs slightly depending on your version control systems.

All commands available from the toolbar are also available from the context menu of a selection.

Item	Tooltip and Shortcut	Description
	Compare Ctrl+D	Click this button to compare the selected revision of a file with its previous revision in the Differences viewer for files .
	Compare with Local	Click this button to compare the selected revision of a file with its local copy in the Differences viewer for files .
	Compare all classes from revision on UML Ctrl+Shift+D	Click this button to view all classes of the selected revision as a UML Class diagram. See section Viewing Changes as Diagram .
	Create Patch	Click this button to create a patch from the selected revision.
	Get	Click this button to retrieve the selected revision. If the local copy has already been modified, IntelliJ IDEA prompts to overwrite the local version, or cancel the operation.
	Annotate	Click this button to open the selected revision of a file in the editor with annotations.
	Show All Affected Files Shift+Alt+A	Click this button to open the Paths Affected in Revision dialog where you can view all files that were modified in the selected revision.
	Copy Revision Number	Click this button to copy the revision number of the commit that the selected file belongs to to the clipboard.
	Show Branches	Click this button to show branches.
	Show All Revisions Submitted In Selected Changelist	Click this button to display the list of all revisions committed in the same changelist as the selected revision of a file.
	Refresh	Click this button to refresh the current information.
	Show Details	Click this button to show the commit message for the selected revision.
	Close Ctrl+Shift+F4	Click this button to close the current history tab.
	Help F1	Click this button to show reference information.

File revisions are shown in regular font. The revision that currently exists in your working copy, is shown in bold font.

Integrate to Branch Info View

View | Tool Windows | Version Control

This view is available after running integration with the Run status after update setting specified. The view displays a list of files or packages affected by the latest integration. The items are displayed under the following nodes:

- Modified
- Merged
- Not in repository
- Locally added

Item	Tooltip and shortcut	Description
		Click this button, to group information within nodes by packages. If the button is released, files are presented in plain lists.
	`Ctrl+NumPad Plus`	Click this button to expand all nodes.
	`Ctrl+NumPad -`	Click this button to collapse all nodes.
		Click this button to close the view.
	`F1`	Click this button to show the corresponding reference page.

Repository and Incoming Tabs

VCS | Show Changes View - Repository/Incoming

View | Tool Windows | Version Control - Repository/Incoming

`Alt+9`

The Repository and Incoming tabs are only available for non-distributed version control systems (i.e. all VCSs supported by IntelliJ IDEA except for Git and Mercurial).

The Repository tab shows the changes committed to the repository under the VCS roots within the current project. The Incoming tab shows the changes committed to the repository by other team members, and not yet checked out locally. Both tabs display the information stored in the history cache. The number of changelists displayed depends on the cache scope .

Each tab contains the following panes:

- The Changelists pane shows changelists.
- The Changed Files pane shows the list of files that were modified and committed within the selected changelist.

Note that if you are using SVN 1.5 or higher both on the server and in the local working copies, the Repository tab also features a Merge Info pane that configures the view in the other two panes and provides control over integration between branches.

Changelists Pane

The pane shows the changelists committed and stored in the history cache. When you click a changelist, the files affected by the selected commit are displayed in the Changed Files pane.

Committed changelists often correspond to issues in tracking systems. You can have such issues opened in the browser right from the Changelists pane. This functionality has the following prerequisites:

- The pattern of the bug tracking system is specified in the Issue Navigation Settings dialog box.
- The corresponding issue number is mentioned in the commit message.

After issue navigation has been configured, issue numbers in commit messages are rendered as links. Clicking such link brings you to the corresponding page of your issue tracker.

Item	Tooltip and Shortcut	Description	Available In
	Refresh `Ctrl+F5`	Click this button to refresh the information in the view.	Both tabs
	Show Details `Ctrl+Q`	Click this button to show the following information on the selected changelist: – Changelist number – User and client name – Date and time of commit	Both tabs
	Create Patch	Click this button to create a patch based on the selected changelist.	Repository tab
	Revert Changes	Click this button to create a *reverse patch* for the selected changelist and roll back the changes made previously. You can use this action to revert changes committed by any user. The Select Target Changelist dialog box opens. Note that if the reverse patch applies to a version committed earlier, this rollback attempt may fail because of the conflicts with the later changes.	Repository tab
	Clear	Click this button to clear the history cache. The commits list will be emptied. To restore it, click Refresh .	Repository
	Edit Revision Comment	Click this button to edit the message for the selected commit.	Repository
	Update Project `Ctrl+T`	Click this button to update the project to the latest available version.	Incoming tab
	Expand All `Ctrl+NumPad Plus`	Click this button to expand all nodes.	Both tabs
	Collapse All `Ctrl+NumPad -`	Click this button to collaps all nodes.	Both tabs
	Copy `Ctrl+C`	Click this button to copy the commit message of the selected changelist to the Clipboard.	Both tabs
	Help `F1`	Click this button to show the corresponding help topic.	Both tabs
	Highlight Integrated	Click this button to have the Merge Info pane displayed. The button is enabled only when both the server side and the client side use Subversion 1.5.	Repository tab

---

Filter by	Use this drop-down list to hide the changelists that are of no interest to you, and only view only the changelists that satisfy a certain criterion. The following options are available: – User: select this option to filter the commits by the user. – Structure: select this option to filter the commits by the target module or folder. – Client: select this option to filter the commits by the computer from which they were made. – None: select this option to turn off filtering and return to the default view.	Both tabs
Group by	Use this drop-down list to group changelists following a certain criterion. The following options are available: – Date: select this option to group the commits by date. – User: select this option to group the commits by users. – Client: select this optio to group the commits by the computer from which they were made.	Both tabs
Search	Use this text box to enter a search pattern and locate the commits whose commit messages matches the specified string. As you type, the list dynamically reduces to show the changelists with the commit messages that match the specified pattern. To save the search pattern, press Enter. To view the list of recent search patterns, click the button. To clear the list of search patterns, click the button.	Repository tab

Changed files pane

Item	Tooltip and Shortcut	Description
	Show Diff Ctrl+D	Click this button to show the differences between the current and the previous revision of the selected file.
	Show Diff with Local	Click this button to show the differences between the selected revision of the selected file and its current local copy.
	Edit Source F4	Click this button to open the local copy of the selected file for editing.
	Open Repository Version	Click this button to open the repository version of the selected file.
	Revert Selected Changes	Click this button to revert the changes to the selected file and roll back to its previous revision.
	Integrate to Branch	Click this button to integrate the changes from the selected file to the target branch.
	Compare Subversion Properties	This option is only available if you are using Subversion as your version control system. Click this button to view the differences in file properties between the current version and the previous revision.
	Show History	Click this button to open the History view of the selected file in the Version Control tool window.
	Group by Directory Ctrl+P	Click this button to transform a flat list of files into a tree of packages with files.
	Expand All Ctrl+NumPad Plus	Click this button to expand all nodes. The button is available only when the files in the pane are displayed grouped by directories.
	Collapse All Ctrl+NumPad -	Click this button to collapse all nodes. The button is available only when the files in the pane are displayed grouped by directories.

Merge Info pane

The pane is available only if you are using SVN 1.5 or higher both on the server and in the local working copies.

In this pane, specify a pair of branches whose integration with each other you want to monitor. The Changelists pane will show the changelists related to the specified branches and provide the information on the integration status of each changelist.

You can specify several pairs of branches if several projects or roots are involved.

Item	Tooltip and Shortcut	Description
From		Specify the URL address of the source branch. IntelliJ IDEA suggests the URL address selected in the Checkout from Subversion dialog box.
To		Do the following: – Specify the path to the target branch.

Click … or press Shift+Enter to open the Select Branch dialog box.

– Specify the path to the local working copy to which you will apply patches created based on the selected changelists.
Click … to open the Configure Working Copy Paths dialog box and select a working copy.

	Highlight Integrated	Click this button to have each changelist in the Changelists pane supplied with an indication of whether it is integrated or not.
	Integrate To Branch	Click this button to integrate the selected changelist into the working copy. The Integrate To Branch dialog box opens.
	Undo Integrate To Branch	Click this button to revert the last integration of the selected changelist into the working copy.
	Mark As Merged	Click this button to indicate that the selected changelist is integrated into the working copy without actually integrating the changelist. The action affects the administrative information in the `.svn` folder. The icon next to the selected changelist changes from to .
	Mark As Not Merged	Click this button to indicate that the selected changelist is not integrated into the working copy without actually reverting integration. Update the administrative information in the `.svn` folder. The icon next to the selected changelist changes from to .
	Filter Out Integrated	Click this button to display only changelists that have not been integrated into the working copy.
	Filter Out Not Integrated	Click this button to display only changelists that have been integrated into the working copy.
	Filter Out Others	Click this button to hide extraneous changelists in the Changelists pane. Extraneous changelists are changelists that are managed in another VCS or are located under another root.
	Show Working Copies	Click this button to open the Subversion Working Copies Information dialog box.
	Refresh	Click this button to refresh the information in the Changelists pane.

Shelf Tab

View | Tool Windows | Version Control - Shelf

N/A

The tab is added to the Version Control tool window when you shelve a change or a changelist, and is displayed until you permanently remove all shelved changes, including the already unshelved ones, and imported external patches.

By default, this tab shows all shelved changes that have not been unshelved yet. Changes are grouped into shelves. A shelf is a changelist created when you shelve changes. A shelf is identified by the commit message. You can have IntelliJ IDEA show the unshelved changes. They can be restored and re-applied as many times as necessary, until they are removed permanently.

For details, see Shelving and Unshelving Changes .

Note You can assign a custom shortcut for the Show Shelf action in Settings | Keymap | Version Control Systems to open the Shelf tab.

Toolbar

Item	Tooltip and shortcut	Description
	Show Diff Ctrl+D	Choose this option to open the Differences viewer for files and compare the shelved version of a file with its current local version.
	Create Patch	Choose this option to create a patch file based on a shelved change. In the Create Patch dialog box that opens, specify the file to save the patch in, and the changes to create a patch from. By default, all changes from the selected shelf are involved in the patch creation. To view which changes are included, click the Selected link. For details, see Creating Patches .
	Show/Hide already unshelved	Click this button to have IntelliJ IDEA show or hide all available shelved changes, both already applied and not. Note that this button is duplicated on the context menu.
		Click this button the reveal the menu of actions described below .

Context menu

The context menu and the toolbar button provide the same set of actions. The context menu is available by right-clicking a change, a change list, or anywhere in the tab.

Item	Shortcut	Description
Unshelve Changes		Choose this option to apply all changes from the selected shelf. In the Unshelve Changes to Changelist dialog box box that opens, specify the changelist you want to add the changes to. For details, see Unshelving Changes .
Unshelve	Ctrl+Shift+U	Choose this option to select specific changes you want to unshelve instead of applying an entire shelf. In the Unshelve Changes to Changelist dialog box box that opens, specify the changes you want to apply, and the changelist you want to add these changes to. For details, see Unshelving Changes .
Restore		Choose this option to re-activate the unshelved changes. By default, unshelved changes are no longer shown in the list of available shelved changes, therefore you first need to have IntelliJ IDEA display it by choosing Show Already Unshelved. You can restore any change as many times as you need until the change is permanently removed by choosing Delete. For details, see Filtering Out and Removing Unshelved Changes and Restoring Unshelved Changes .
Import Patches		Choose this option to apply patches created externally or through IntelliJ IDEA. In the dialog box that opens, choose the files to import patches from. The imported patches are treated as shelved changes and are shown in the Shelf tab of the Version Control tool window where you can unshelve them at any time.
Rename	Shift+F6	Choose this option to modify the name of the selected shelved changelist.
Delete	Delete	Choose this option to permanently delete the selected shelved change. Tiy can remove any change no matter whether it has been already unshelved or not. For details, see Filtering Out and Removing Unshelved Changes .
Show Already Unshelved		Choose this option to have IntelliJ IDEA show all available shelved changes, both already applied and not. By default, unshelved changes are hidden. For details, see Filtering Out and Removing Unshelved Changes .
Delete All Already Unshelved		Choose this option to have IntelliJ IDEA permanently remove all shelved changes that have been already applied. For details, see Filtering Out and Removing Unshelved Changes .

Update Info Tab

View | Tool Windows | Version Control

This tab is available when local information is synchronized to the server.

Item	Tooltip and shortcut	Description
	Group by Packages	When this button is pressed, the update information within nodes is grouped by packages.
	Group by Changelists	When this button is pressed, the update information within nodes is grouped by changelists, and by the day the changelist has been committed. The Update Info tab is divided into two panes: the left pane shows the changelists, grouped by the check-in date, and the right pane shows the list of changed files . Grouping by changelists is not available if the project is under Git or Mercurial control.
	Expand all Ctrl+NumPad Plus	Use these buttons to have all nodes expanded or collapsed.
	Collapse all Ctrl+NumPad -	
	Show Diff Ctrl+D	Click this button to open the Differences viewer for files , where you can compare the local copies of all the project files one after another with their updates from the server. Use the buttons Compare Next File and Compare Previous File to scroll through the list of updated files.
	Close Ctrl+Shift+F4	Click this button to close the tab.
	Help F1	Click this button to show reference page.
The controls of the Changed files pane appear when the button is pressed.		
	Show Diff Ctrl+D	Click this button to show differences between the selected and the previous revision for the selected file in the Changes Files pane.
	Show Diff with Local	Click this button to show differences between the selected revision of a file, and its current local copy.
	Edit Source F4	Click this button to open local copy of the selected file for editing.
	Open Repository Version	Click this button to open repository version of the selected file.
	Revert Selected Changes	Click this button to revert selected changes.
		This button is only available for the files under Subversion, and enables viewing differences in properties of the selected file.
	Show History	Click this button to open the History view of the selected file in the History Tab of the Version Control tool window.
	Group by Directory Ctrl+P	When the button is not pressed, the pane shows a flat list of files. When the button is pressed, the pane shows files in their respective packages. In the latter case, expand and collapse buttons appear in the toolbar.
	Expand all Ctrl+NumPad Plus	Use these buttons to have all nodes expanded or collapsed.
	Collapse all Ctrl+NumPad -	
	Select All Ctrl+A	Click this button to select all files in the Changed Files pane.

MQ: <project_name> Tab

View | Tool Windows | Version Control - Log - Context menu of a commit - Mercurial - Goto patch

This tab is only available if your project is under Mercurial version control.

It displays all patches and allows you to manage the Mercurial Queue .

You can drag-and-drop patches in the queue to change the order in which they will be applied.

Toolbar and Context Menu

Item	Shortcut	Description	Available from
Reload from file	Ctrl+F5	Use this command to refresh the list of patches from the `series` file. Note that when you drag-and-drop patches in the queue, the corresponding changes in the `series` file will only be saved when you switch to a different tab, or perform some other external action. This allows you to revert such changes simply by reloading from file.	Toolbar
Goto	Shift+Alt+G	Use this command to apply the selected patch and all patches above it in the queue. The applied patches will become visible in the Log tab and the changes will be registered in the working copy of the repository.	Toolbar Context menu
Move and Push	Shift+Alt+P	Use this command to apply the selected patch without applying whatever other patches may be before it in the patch stack.	Toolbar Context menu
Fold	Shift+Alt+D	Use this command to merge the selected non-applied patch with the topmost applied patch, and remove it from the list.	Toolbar Context menu
Delete	Delete	Use this command to delete the selected patch from the series file	Toolbar
Apply Patch	N/A	Use this command to apply the selected patch. The Apply Patch dialog will be displayed where you can select which changes you want to restore.	Context menu

V8 Heap Tool Window

The tool window opens when you take a snapshot and choose to open it. The tool window shows the collected profiling data. If the window is already opened and shows the profiling data for another session, a new tab is added. Tabs that were opened automatically are named after the run configurations that control execution of the applications and collecting the profiling data.

If you want to open and analyze some previously saved mempry profiling data, choose V8 Profiling - Analyze V8 Heap Snapshot on the main menu and select the relevant `.snapshot` file. IntelliJ IDEA creates a separate tab with the name of the selected file.

The tool window has three tabs that present the collected information from difference point of views.

On this page:

– Containment
– Biggest Objects
– Summary
– Details Pane
– Toolbar
– Context Menu of an Object

Containment

The tab shows the objects in you application grouped under several top-level entries: DOMWindow objects, Native browser objects, and GC Roots, which are roots the Garbage Collector actually uses. See Containment View for details.

For each object, the tab shows its distance from the GC root, that is the shortest simple path of nodes between the object and the GC root, the shallow size of the object, and the retained size of the object. Besides the absolute values of the object's size, IntelliJ IDEA shows the percentage of memory the object occupies.

Biggest Objects

The tab shows the most memory-consuming objects sorted by their retained sizes . In this tab, you can spot memory leaks provoked by accumulating data in some global object.

Summary

The tab shows the objects in your application grouped by their types. The tab shows the number of objects of each type, their size, and the percentage of memory that they occupy. This information may be a clue to the memory state.

Details Pane

Each tab has a Details pane, which shows the path to the currently selected object from GC roots and the list of object's retainers, that is, the objects that keep links to the selected object. Every heap snapshot has many "back" references and loops, so there are always many retainers for each object.

Navigating through a Snapshot

– To help differentiate objects and move from one to another without losing the context, mark objects with text labels. To set a label to an object, select the object of interest and click on the toolbar or choose Mark on the context menu of the selection. Then type the label to mark the object with in the dialog box that opens.

– To navigate to the function or variable that corresponds to an object, select the object of interest and click on the toolbar or choose Edit Source on the context menu of the selection. If the button and the menu option are disabled, this means that IntelliJ IDEA has not found a function or a variable that corresponds to the selected object.
If several functions or variables are found, they are shown in a pop-up suggestion list.

– To jump from an object in the Biggest Objects or Summary tab or Occurrences view to the same object in the Containment tab, select the object in question in the Biggest Objects or Summary tab and click on the toolbar or choose Navigate in Main Tree on the context menu of the selection. This helps you investigate the object from the containment point of view and concentrate on the links between objects.

– To search through a snapshot:
1. In the Containment tab, click on the toolbar.
2. In the V8 Heap Search Dialog that opens, specify the search pattern and the scope to search in. The available scopes are:
   – Everywhere: select this check box to search in all the scopes. When this check box is selected, all the other search types are disabled.
   – Link Names: select this check box to search among the object names that V8 creates when calling the C++ runtime, see http://stackoverflow.com/questions/11202824/what-is-in-javascript .
   In the V8 Heap Tool Window , link names are marked with the `%` character (`%<link name>`).
   – Class Names: select this check box to search among functions-constructors.
   – Text Strings: select this check box to perform a textual search in the contents of the objects.
   – Snapshot Object IDs: select this check box to search among the unique identifiers of objects. V8 assigns such a unique identifier in the format to each object when the object is created and preserves it until the object is destroyed. This means that you can find and compare the same objects in several snapshots taken within the same session.
   In the V8 Heap Tool Window , object IDs are marked with the `@` character (`@<object id>`).
   – Marks: select this check box to search among the labels you set to objects manually by clicking on the toolbar of the Containment tab.

   The search results are displayed in the Details pane, in a separate Occurrences of '<search pattern>' view. To have the search results shown grouped by the search scopes you specified, press the Group by Type toggle button on the toolbar.

   When you open the dialog box next time, it will show the settings from the previous search.

Toolbar

The toolbar is common for all tabs and most of the toolbar buttons are available in all tabs.

ItemDescriptionAvailable

	in	
	Click this button to set a label to the selected object. This helps you differentiate objects and move from one to another without losing the context.	Containment tab, Biggest Objects tab, Summary tab, Details pane, Occurrences view
	Click this button to find a function in a snapshot: in the V8 Heap Search Dialog that opens, specify the search pattern and the scope to search in. The available scopes are: – Everywhere: select this check box to search in all the scopes. When this check box is selected, all the other search types are disabled. – Link Names: select this check box to search among the object names that V8 creates when calling the C++ runtime, see http://stackoverflow.com/questions/11202824/what-is-in-javascript . In the V8 Heap Tool Window , link names are marked with the `%` character (`%<link name>`). – Class Names: select this check box to search among functions-constructors. – Text Strings: select this check box to perform a textual search in the contents of the objects. – Snapshot Object IDs: select this check box to search among the unique identifiers of objects. V8 assigns such a unique identifier in the format to each object when the object is created and preserves it until the object is destroyed. This means that you can find and compare the same objects in several snapshots taken within the same session. In the V8 Heap Tool Window , object IDs are marked with the `@` character (`@<object id>`). – Marks: select this check box to search among the labels you set to objects manually by clicking on the toolbar of the Containment tab.	Containment tab
	Click this button to jump from an object in the Biggest Objects or Summary tab or Occurrences view to the same object in the Containment tab. This helps you investigate the object from the containment point of view and concentrate on the links between objects.	Biggest Objects tab, Summary tab, Occurrences view
	Click this button to navigate to the function or variable that corresponds to the selected object.	Containment tab, Occurrences view
	Press this toggle button to have the search results shown grouped by the search scopes you specified.	Occurrences view
	Click this button to open the reference page for the tool window.	All
	Click this button to close the tool window.	All

Context Menu of an Object

Item	Description
Mark	Choose this option to set a label to the selected object. This helps you differentiate objects and move from one to another without losing the context.
Navigate in Main Tree	Choose this option to jump from an object in the Biggest Objects or Summary tab or Occurrences view to the same object in the Containment tab. This helps you investigate the object from the containment point of view and concentrate on the links between objects.
Jump to Source	Choose this option to navigate to the function or variable that corresponds to the selected object.

V8 Heap Search Dialog

The dialog box opens when you click on the toolbar in the V8 Heap Tool Window Use the dialog box to find constructors, object IDs, character strings, etc. in a snapshot. The search results are displayed in the Details pane, in a separate Occurrences of '<search pattern>' view.

Item	Description
Search	In this text box, type the search pattern to look for. Select the Case Sensitive check box if necessary.
Scope	In this area, specify the type of objects to limit the search to. When the Everywhere check box is selected, all the other search types are not available. – Everywhere: select this check box to search in all the scopes. When this check box is selected, all the other search types are disabled. – Link Names: select this check box to search among the object names that V8 creates when calling the C++ runtime, see http://stackoverflow.com/questions/11202824/what-is-in-javascript . In the V8 Heap Tool Window , link names are marked with the `%` character (`%<link name>`). – Class Names: select this check box to search among functions-constructors. – Text Strings: select this check box to perform a textual search in the contents of the objects. – Snapshot Object IDs: select this check box to search among the unique identifiers of objects. V8 assigns such a unique identifier in the format to each object when the object is created and preserves it until the object is destroyed. This means that you can find and compare the same objects in several snapshots taken within the same session. In the V8 Heap Tool Window , object IDs are marked with the `@` character (`@<object id>`). – Marks: select this check box to search among the labels you set to objects manually by clicking on the toolbar of the Containment tab.

V8 Profiling Tool Window

The tool window is opened automatically when you stop the Node.js application you are profiling. If the window is already opened and shows the profiling data for another session, a new tab is added. Tabs that were opened automatically are named after the run configurations that control execution of the applications and collecting the profiling data.

If you want to open and analyze some previously saved profiling data, choose V8 Profiling - Analyze V8 Profiling Log on the main menu and select the relevant V8 log file `isolate-<session number>`. IntelliJ IDEA creates a separate tab with the name of the log file.

Based on the collected profiling data, IntelliJ IDEA builds three call trees and displays each of them in a separate pane. Having several call trees provides the possibility to analyze the application execution from two different points of view: on the one hand, which calls were time consuming ("heavy"), and on the other hand, "who called whom".

On this page:

– Toolbar
– Context Menu
– Top Calls Pane
– Bottom-up Pane
– Top-down Pane
– Flame Chart
– Selecting a Fragment in the Timeline
– Synchronization in the Flame Chart

Toolbar

Item	Tooltip	Description	Available in
	Jump to source	Click this button to navigate to the function definition.	Top Calls Bottom-up Top-down
	Filter	Click this button to filter out light calls and have IntelliJ IDEA display only the calls that indeed cause performance problems. Using the slider, specify the minimum Total% or Parent% value for a call to be displayed and click Done.	Top Calls Bottom-up Top-down
	Expand Heavy Traces	When a tab for a profiling session is opened, by default the nodes with heaviest calls are expanded. While exploring the trees, you may like to fold some nodes or expand other ones. Click this button to restore the original tree presentation.	Top Calls Bottom-up Top-down
 	Expand All/ Collapse All	Click these buttons to expand or collapse all the nodes in the current pane.	Top Calls Bottom-up Top-down
	Export to text file/ Export Timeline Chart	Click this button to save the call tree in the current pane to a text file or the current timeline chart to a `.png` file. Then specify the target file in the dialog box that opens.	All
	Help	Click this button to navigate to the Help topic for the tool window.	All
	Close	Click this button to close the V8 Profiling tool window.	All
	Zoom	Click this button to open the selected fragment of the flame chart in a separate tab and have the selected fragment enlarged to fit the tab width so you can examine the fragment with more details.	Flame Chart

Context Menu

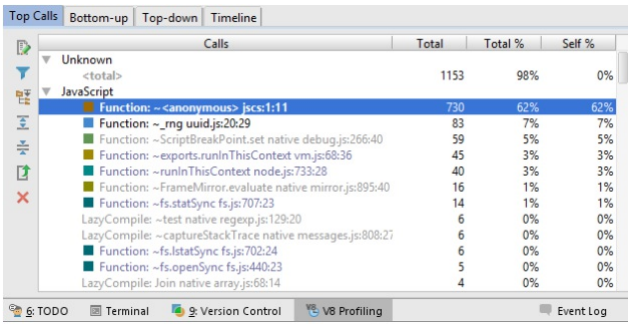
The context menu is available only from items in the Top Calls, Bottom-up, and Top-down panes.

Item	Description
Copy Call	Choose this option to copy the name of the selected function and the name of the file where it is defined to the Clipboard.
Copy	Choose this option to copy the name of the selected function, the name of the file where it is defined, and the measurements data. This may be helpful if you want to compare the measurements for a function from two sessions, for example, after you make some improvements to the code.
Compare with Clipboard	Choose this option to compare the selected with the contents of the Clipboard in the Difference Viewer that opens.

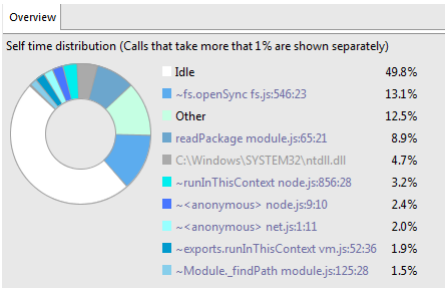
Expand Node/ Collapse Node — Choose these options to expand or collapse the selected tree node.

Top Calls Pane

The Top Calls pane shows a list of performed activities sorted in the descending order by the Self metrics. For each activity IntelliJ IDEA displays its Total, Total%, and Self% metrics. For each function call, IntelliJ IDEA displays the name of the file, the line, and the column where the function is defined.



The diagram in the Overview pane shows distribution of self time for calls with the Self% metrics above 1%.



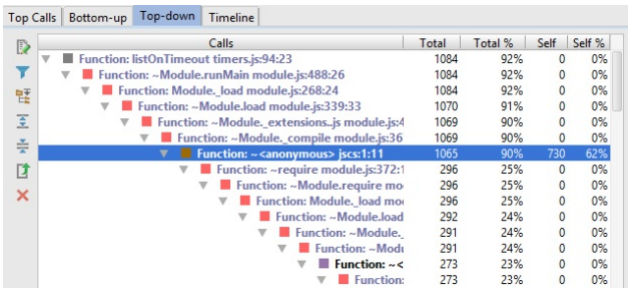
Bottom-up Pane

The Bottom-up pane also shows the performed activities sorted in the descending order by the Self metrics. Unlike the Top Calls pane, the Bottom-up pane shows only the activities with the Total% metrics above 2 and the functions that called them. This is helpful if you encounter a heavy function and want to find out where it was called from. For each activity IntelliJ IDEA displays its execution time in ticks and the Of Parent metrics. For each function call, IntelliJ IDEA displays the name of the file, the line, and the column where the function is defined.

Top-down Pane

The Top-down pane shows the entire call hierarchy with the functions that are execution entry points at the top. For each activity IntelliJ IDEA displays its Total, Total%, Self, and Self% metrics. For each function call, IntelliJ IDEA displays the name of the file, the line, and the column where the function is defined. Some of the functions may have been optimized by V8, see Optimizing for V8 for details.

- The functions that have been optimized are marked with an asterisk (*) before the function name.
- The functions that possibly require optimization but still have not been optimized are marked with a tilde (~) character before the function name. Though optimization may be delayed by the engine or skipped if the code is short-running, a tilde (~) points at a place where the code can be rewritten to achieve better performance.



Flame Chart

Use the multicolor chart in the Flame Chart tab to find where the application paused and explore the calls that provoked these pauses. The chart consists of four areas:

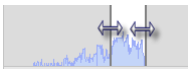
- The upper area shows a timeline with two sliders to limit the beginning and the end of a fragment to investigate.
- The bottom area shows a stack of calls in the form of a multicolor chart. When called for the first time, each function is assigned a random color, whereupon every call of this function within the current session is shown in this color.
- The middle area shows a summary of calls from the Garbage Collector, the engine, the external calls, and the execution itself. The colors reserved for the Garbage Collector, the engine , the external calls, and the execution are listed on top of the area:

- The right-hand pane lists the calls within a selected fragment, for each call the list shows its duration, the name of the called function, and file where the function is defined.

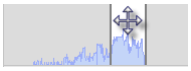
Selecting a Fragment in the Timeline

To explore the processes within a certain period of time, you need to select the fragment in question. You can do it in two ways:

- Use the sliders:



– Click the window between two sliders and drag it to the required fragment:



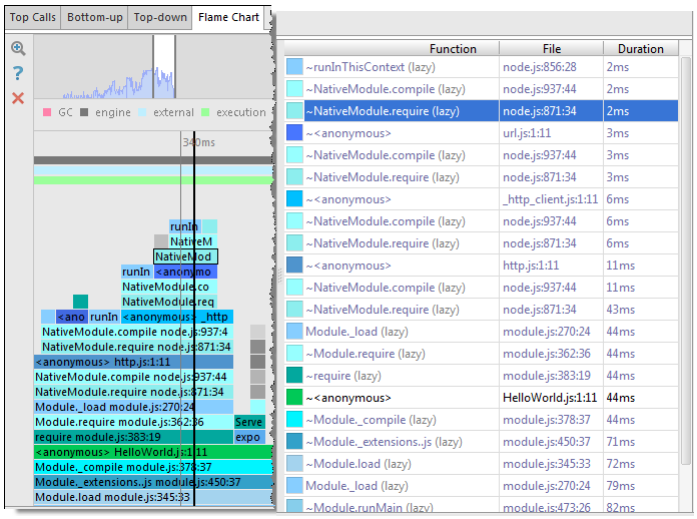
In either case, the multicolor chart below shows the stack of calls within the selected fragment.

To enlarge the chart, click the selected fragment and then click the Zoom button on the toolbar. IntelliJ IDEA opens a new tab and shows the selected fragment enlarged to fit the tab width so you can examine the fragment with more details.

Synchronization in the Flame Chart

The bottom and the right-hand areas are synchronized: as you drag the slider in the bottom area through the timeline the focus in the right-hand pane moves to the call that was performed at each moment.

Moreover, if you click a call in the bottom area, the slider moves to it automatically and the focus in the right-hand pane switches to the corresponding function, if necessary the list scrolls automatically. And vice versa, if you click an item in the list, IntelliJ IDEA selects the corresponding call in the bottom area and drags the slider to it automatically:



IntelliJ IDEA supports navigation from the right-hand area to the source code of called functions, to the other panes of the tool window, and to areas in the flame chart with specific metrics.

– To jump to the source code of a called function, select the call in question and choose Jump to Source on the context menu of the selection.
– To switch to another pane, select the call in question, choose Navigate To on the context menu of the selection, and then choose the destination:
  – Navigate in Top Calls
  – Navigate in Bottom-up
  – Navigate in Top-down

  IntelliJ IDEA switches to the selected pane and moves the focus to the call in question.
– To have the flame chart zoomed at the fragments with specific metrics of a call, select the call in question, choose Navigate To on the context menu of the slection, and then choose the metrics:
  – Navigate to Longest Time
  – Navigate to Typical Time
  – Navigate to Longest Self Time
  – Navigate to Typical Self Time

You can also navigate to the stacktrace of a call to view and anlyze exceptions. To do that, select the call in question and choose Show As Stacktrace. IntelliJ IDEA opens the stacktrace in a separate tab, to return to the Flame Chart pane, click V8 CPU Profiling tool window button in the bottom tool window.

SBT Tool Window

View | Tool Windows | SBT

The SBT tool window displays SBT tasks, and all changes made to the underlying `build.sbt` file, or to the IntelliJ IDEA project structure.

If there is no linked SBT project, the window is disabled. You need to reimport your SBT project to enable the SBT tool window.

In this section:

– SBT Projects
– Toolbar
– Context Menu

SBT Projects

Item	Description
	Click this button to open the menu for configuring the current view and changing the tool window viewing modes. The menu items are options that you can turn on or off. An active item contains a check mark on the left side of its name. The available options in the menu are as follows: – Group Tasks - select this option if you want to group your tasks. For example, build tasks will be included into the build directory. – Show Toolbar - select this option to show the toolbar for your Gradle projects. – Pinned, Docked, Floating, Windowed, Split Mode – Group Tabs - deselect this option to see the views on separate tabs if more than one view is available in a tool window. – Move to - select this option to move the Gradle tool window to either top, left or right. – Resize - select this option to resize the Gradle tool window.
	Click this button to hide the tool window(`Shift+Escape`). When used in combination with the Alt key, clicking this button hides all the tool windows attached to the same tool window bar.

Toolbar

Item	Description
	Click this button to refresh all registered Gradle projects after changes have been made to a Gradle script.
	Click this button to link an sbt project.
	Click this button to detach an external sbt project.
	Use these buttons to expand or collapse all the nodes.
	Click this button to import a module or a data to your sbt project. It might be useful for the multi-module projects.
	Click this button to configure the settings of the current SBT project.

Context Menu

Item	Description
Open SBT config	Select this option to open `sbt.build` file in the editor.
Refresh external project	Select this option to refresh all external projects.
Detach external project	Select this option to detach an external project.
Ignore SBT project	If you select this option, IntelliJ IDEA starts to ignore the selected module on the next project's import.
Task Activation	Select this option to activate tasks to run.

Web Tool Window

View | Tool Windows | Web

The Web tool window presents the project structure as a tree where only modules with Web facets are shown. Below each module node, the corresponding `web.xml` deployment descriptor is shown.

Use this tool window to:

- Find out which modules have Web facets.
- Configure elements in Web applications: servlets, filters, and listeners.

Context Menu

Item	Description
Jump to Source (F4)	Select this option: - To open the selected file in the editor. (To open a file in the editor, you can, alternatively, double-click the file.) - To jump to the selected folder in the Project tool window.
New	Select this option to create a Web application element . Choose the element type in the submenu. The available options are: - Servlet - Filter - Listener

Import Project or Module Wizard

From the Welcome screen (if no project is currently open): Import Project

File | New | Project from Existing Sources

File | New | Module from Existing Sources

From the Project Structure dialog: Modules | + | Import Module

---

Use this wizard to import existing sources into IntelliJ IDEA and, as a result, to create a new project (New | Project from Existing Sources) or to add a module (or a number of modules) to an existing project (New | Module from Existing Sources).

- Origin of the Sources
- Import Existing Sources. Project Name and Location
- Import Existing Sources. Source Root Directories
- Import Existing Sources. Libraries
- Import Existing Sources. Module Structure
- Import Existing Sources. Project SDK
- Import Existing Sources. Frameworks
- Import from Bnd/Bndtools. Page 1
- Import from Eclipse. Page 1
- Import from Eclipse. Page 2
- Import from Flash Builder. Page 1
- Import from Flash Builder. Page 2
- Import Project from Gradle. Page 1
- Import from Maven. Page 1
- Import from Maven. Page 2
- Import from Maven. Page 3
- Import from Maven. Page 4
- Import Project from SBT. Page 1

Origin of the Sources

If the sources that you are importing come from Bnd/Bndtools , Eclipse , Flash Builder , Gradle , or Maven , select Import project from external model or Import module from external model, and select Bnd/Bndtools, Eclipse, Flash Builder, Gradle, or Maven respectively.

Otherwise, select Create project from existing sources or Create module from existing sources.

Import Existing Sources. Project Name and Location

Specify the project name and location.

Item	Description
Project name	Specify the name of the project to be created.
Project location	Specify the path to the directory in which your sources, libraries and other assets of interest are located. You can click ... (Shift+Enter) and select the necessary directory in the dialog that opens . Note that the specified directory will act as the project root directory. (In technical terms, this is where the `.idea` directory will be created. For more information, see Project format and files and Directory-based format.)

Import Existing Sources. Source Root Directories

This page shows the directories in which source files are found. Potentially, these directories may be your source root directories .

Use the check boxes to select the directories which you want to include in your project as the source root directories .

Item	Description
Mark All	Use this button to select all the directories.
Unmark All	Use this button to deselect all the directories.

Import Existing Sources. Libraries

This page lets you select and configure the libraries to be included in your project.

In the left-hand pane, under Libraries, the list of libraries and archives found in the specified location are shown. When you highlight an item in this pane, its contents are shown in the right-hand pane under Library contents.

Use the check boxes to select the libraries which you want to include in your project. Use other controls available on this page to rename the libraries, and also split or merge them (see below).

Item	Description
	Click this icon to rename the selected (highlighted) library or archive.
	Click this icon to join several selected (highlighted) libraries or archives into a new library.
	Click this button to split the selected (highlighted) library into two parts. In the Split Library dialog, select the files to be extracted into the new library.

Import Existing Sources. Module Structure

This page lets you select and configure the modules to be included in your project.

In the left-hand pane, under Modules, the modules that potentially may be included in your project are shown. When you highlight a module in this pane, its dependencies are shown in the right-hand pane under Module dependencies. (If you highlight a number of modules, their aggregated dependencies are shown.)

Use the check boxes to select the modules which you want to include in your project. Use other controls available on this page to rename the modules, and also split or merge them (see below).

Item	Description
	Click this icon to rename the selected (highlighted) module.
	Click this icon to join several selected (highlighted) modules into a new one.
	Click this icon to split the selected (highlighted) module into two parts. In the Split Module dialog, select the content roots to be assigned to the new module.

Import Existing Sources. Project SDK

Specify the SDK to be used in your project .

If the necessary SDK is already defined in IntelliJ IDEA, select it in the pane under + and — (in the left-hand part of the page).

Otherwise, click + and select the SDK type . Then, in the dialog that opens , select the installation folder of the desired SDK. (By this time, the corresponding SDK must already be installed on your computer. If it isn't, download and install it first.)

Import Existing Sources. Frameworks

If within the specified location IntelliJ IDEA finds the files that are indicators of certain frameworks and technologies, you can choose to enable support for these frameworks and technologies. To select the necessary frameworks and technologies, use the corresponding check boxes.

Frameworks and technologies in IntelliJ IDEA are represented by facets .

Item	Description
Group by	Specify how the files-indicators should be grouped: – Type. The files are grouped by frameworks and technologies. The files that belong to the same technology appear in the same group. Group names correspond to the names of the frameworks and technologies or, in IntelliJ IDEA terms, facet types (Web, JSF, Spring, etc.). – Directory. The files are grouped by their locations. The files located in the same directory appear in the same group.

Import from Bnd/Bndtools. Page 1

Use this page to select a Bnd/Bndtools project to import.

Item	Description
Select Bnd/Bndtools projects to import	Use this area to select a Bnd/Bndtools project that you want to import.
Select all	Click this button to select all the check boxes next to Bnd/Bndtools projects for importing.
Unselect all	Click this button if you want to clear all the check boxes next to Bnd/Bndtools projects.
Open Project Structure after import	Select this check box to open the Project Structure dialog right after you import your project.

Import from Eclipse. Page 1

Use this page to define the source workspace and the target location for the imported Eclipse projects.

Item	Description
Select Eclipse project directory	Specify the path to the Eclipse workspace that contains projects to be imported. You can type the path, or click the ellipsis button and locate the desired directory in the Select Path dialog.
IntelliJ IDEA project and module files location	**Option** / **Description** Create module files near .classpath files: If you select this option, an IntelliJ IDEA module per each Eclipse project will be created in the respective projects directory; the IntelliJ IDEA project in the specified format will be created in the root of the Eclipse workspace, or Eclipse project directory. Keep project and module files in: If you select this option, enter the target location of the `*.iml` files to be created from the imported Eclipse projects, or click the ellipsis button, and navigate to the desired location.
Project file format	Choose file-based or directory-based format of saving project.
Link created IntelliJ IDEA modules to Eclipse project files	Check this option to automatically keep the Eclipse projects and IntelliJ IDEA modules synchronized.
Detect test sources	Specify the list of roots, where test sources should be sought for. Refer to the Compiler section for the wildcards syntax.

Import from Eclipse. Page 2

Use this page to browse the projects detected in the selected Eclipse workspace and choose the ones to be imported to the IntelliJ IDEA project.

Item	Description
Select Eclipse projects to import	In the list of projects detected in the specified root, select the ones to be imported. Each Eclipse project will be converted into an IntelliJ IDEA module.
Open Project Structure after import	Select this check box to have the Project Structure dialog box opened when the import is completed.

Import from Flash Builder. Page 1

Specify the Flash Builder projects to be imported and associated settings.

Item	Description
Flash Builder workspace or project...	Specify the location of a file or directory that you want to use as a source of import. This can be: – A Flash Builder workspace or project directory – A `.project` file – A `.fxp` or `.fxpl` file – A `.zip` file that contains an ActionScript project or projects. Type the path to the necessary file or directory right in the field, or click … and select the desired file or folder in the dialog that opens .
Project name	When creating a project: specify the name of the IntelliJ IDEA project to be created.
Project location	When creating a project: specify the project location. (This is where the `.idea` directory or a `.ipr` file will be created, see Project format and files. Besides, if you are importing an archive (`.fxp` , `.fxpl` , or `.zip`), the archive will be extracted into this folder or its subfolder.) Edit the path in the field or click … and select the desired folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)
Create subfolder...	When creating a project: this option is available if there is only one project within the Flash Builder source that you are importing. In such a case, you have an option of creating a subfolder in the project folder and placing the imported project there. If the source contains two or more projects, the individual subfolders will be created for each of them.
Project file format	When creating a project: if necessary, change the suggested project format .
Extract project to or Multiple projects found...	When importing into an existing project: if you are importing an archive (`.fxp` , `.fxpl` , or `.zip`), you can specify where the archive should be extracted. If the source contains two or more projects, individual subfolders in the specified location will be created for each of them. Edit the path in the field or click … and select the desired folder in the dialog that opens . (You can create a new folder in that dialog, e.g. by using .)

Import from Flash Builder. Page 2

If you are importing a Flash Builder workspace, this page shows the projects detected therein. Select the projects that you want to import.

Item	Description
Select Flash Builder projects to import	Select the projects that you want to import.
Open Project Structure after import	Select this check box to have the Project Structure dialog opened when the import is completed.

Import Project from Gradle. Page 1

Item	Description
Gradle project	Specify the address of the `build.gradle` file of the project you want to import. Type the path manually, or click the browse button … and locate the desired file in the file chooser dialog.
Create directories for empty content roots automatically	Select this option to add a `src` directory to your project automatically when you import a project from Gradle model.
Create separate module per source set	Select this check box to use the source set feature in resolving your Gradle projects.
Use default gradle wrapper (not configured for the current project)	Select this option to use Gradle wrapper. Using Gradle wrapper lets you get automatic Gradle download for the build. It also lets you build with the precise Gradle version.
Use customizable gradle wrapper	Select this option to configure a Gradle wrapper inside your `build.gradle` file. In this case IntelliJ IDEA ensures that the configured customizable wrapper settings are used for working with Gradle.
Use local gradle distribution	Select this option to run local build scripts.
Gradle home	By default, IntelliJ IDEA displays the path to the Gradle installation, as defined in the `GRADLE_HOME` environment variable, or specified as the Gradle home project setting . If the path is unknown, or you want to use a different Gradle installation, click the browse button … and locate the desired directory in the path chooser dialog.
Project format	Select the format in which you want to store your project. For the details on the two available formats, refer to the Project section.
Global Gradle Settings	Use this area to set global Gradle settings. You can select from the following options: – Offline work - use this check box to work with Gradle in the offline mode. In this case Gradle will use dependencies from the cache. Gradle will not attempt to access the network to perform dependency resolution. If required dependencies are not present in the dependencies' cache, a build execution will fail. – Service directory path - use this field to override the default Gradle home location directory. – Gradle VM options - use this field to specify VM options for your Gradle project. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .

Import from Maven. Page 1

Use this page to specify the import preferences. The default import settings are defined in the Maven Integration dialog .

Item	Description
Root directory	Specify the directory where the Maven projects you want to import are located.
Search for projects recursively	Select this check box to make import look for the nested `pom.xml` files.
Project format	Select the format in which you want to store your project. For the details on the two available formats, refer to the Project section.
Keep project files in	Select this check box to specify the desired location for storing project and module files.
Import Maven projects automatically	Select this check box if you want IntelliJ IDEA to perform reimport automatically each time you change your `pom.xml` .
Create IntelliJ IDEA modules for aggregator projects (with 'pom' packaging)	Clear this check box to skip creating IntelliJ IDEA modules. You might want to do that if the Maven project you import is an aggregator, and its `packaging` element has the value `pom` .
Create module groups for multi-module Maven projects	If this check box is selected, IntelliJ IDEA will create a module group from an aggregative Maven project, with the nested modules included in this group.
Keep source and test folders on reimport	If this check box is selected, the source and test folders are kept when you reimport a project.
Exclude build directory (PROJECT_ROOT/target)	Clear this check box to disable the automatic exclusion of target folders. Note if a `/target` directory contains `/target/generated-sources` , then the `/target` directory cannot be excluded since `/target/generated-sources` directory is marked as `sources` . Other subdirectories in the `/target` directory are excluded.
Use Maven output directories	If this check box is not selected, the build is created in the regular IntelliJ IDEA's output directory `USER_HOME\IdeaProjects\<project>\classes\Production\` . If this check box is selected, the build is generated in the Maven's output directory, and the results of IntelliJ IDEA's compilation are reused. However, IntelliJ IDEA itself does not reuse Maven build results and performs compilation *from scratch*.
Generated sources folders	Specify the directory of your source root when you reimport a project. You can select one of the following options: − Detect automatically This is a default option. When you select this option, IntelliJ IDEA automatically detects the location of the generated sources. IntelliJ IDEA also detects which directory to mark as a source root. Note that IntelliJ IDEA searches for the generated sources only in `target/generated-sources` and `target/generated-sources/*` directories. − target/generated-sources This option enables you to mark the directory as a source root manually. − subdirectories of "target/generated-sources" This option enables you to mark a subdirectory as a source root manually.
Phase to be used for folders update	Select Maven phase to be used for folders update. This might be useful if you adjust your plugins so that the additional sources are loaded at some phase.
Automatically download	Specify the automatic download of sources and documentation comments into a local repository when you open a Maven project. You can select one or both of the following check boxes: − Sources − Documentation
Dependency types	Use this field to specify dependency types that you want to include in your import.
Environment settings	Click this button to specify Maven Environment settings.

Maven Environment Dialog

Use this page to specify Maven environment settings.

Item	Description
Maven home directory	Use this drop-down list to select a bundled Maven version that is available (for Maven2, version 2.2.1 and for Maven3, version 3.0.5) or the result of resolved system variables such as `MAVEN_HOME` or `MAVEN2_HOME`. You can also specify your own Maven version that is installed on your machine. You can click … and select the necessary directory in the dialog that opens.
User settings file	Specify the path to the file that contains user-specific configuration for Maven.
Local repository	By default, the field shows the path to the local directory under the user home that stores downloads and contains the temporary build artifacts that you have not yet released.
Override	Select this check box next to a field where you need to specify a different directory, click the ellipsis button and select the desired path.

Import from Maven. Page 2

Use this page of the wizard to select the Maven profiles to be applied to the new project.

Item	Description
Select profiles	Check the build profiles to be applied to the new project.

Import from Maven. Page 3

Use this page to select the Maven project to be imported into the new IntelliJ IDEA project.

Item	Description
Select Maven projects to import	In the list of Maven projects, detected in the specified root, check the projects to be imported.
Open project structure after import	Check this option to open the Project Structure dialog after completing the import.

Import from Maven. Page 4

Use this page to create a name for your new IntelliJ IDEA project.

Item	Description
Project name	Specify the name of your IntelliJ IDEA project.
Project file location	Specify the location of your project directory where IntelliJ IDEA stores the configuration data for your project and its components. You can use ellipsis button to navigate to the appropriate directory.

Import Project from SBT. Page 1

Use this page to specify SBT import project settings.

Item	Description
SBT project	Specify the name of the project that you import.
Use auto-import	Select this check box to resolve all changes made to the SBT project automatically every time you refresh your project.
Create directories for empty content roots automatically	Select this check box to add an `src` directory to your project.
Download sources and docs	Select this check box to download dependencies and their sources for your project.
Download SBT sources and docs	Select this check box to download SBT dependencies and their sources for your project. The dependencies are used in build files such as `build.sbt`.
Project SDK	Use this field to specify SDK to be used in your project. If the project SDK is already defined in IntelliJ IDEA, select it from the list. Otherwise, click New and select the installation folder of the desired IntelliJ IDEA version.
Global SBT settings	Use this area to set global SBT settings.
JVM	Use this area to set JVM. You can choose from the following options: – From project JDK - use this option if you want to use a default JVM from your project's JDK. – Custom - use this option if you want to use a custom JVM.
JVM Options	Use this area to set the following JVM options: – Maximum heap size, MB - use this field to specify the maximum heap size available to the process that launches the compiler. The default 768 Mb is suitable for most of the purposes. – VM parameters - use this field to type the string to be passed to the VM when IntelliJ IDEA launches the compiler. If you need more room to type, click to open the VM parameters dialog where the text entry area is larger.
Launcher (sbt-launch.jar)	Use this area to specify the SBT launcher. You can choose from the following options: – Bundled - select this option if you want to use a default launcher located the plugin directory. The latest version of the launcher is usually available. – Custom - select this option if you want to use a custom launcher. Specify the path to the location of your custom launcher.

Dialogs

This part contains descriptions of the following dialogs:

- Add Composer Dependency Dialog
- Add Frameworks Support Dialog
- Add New Field or Constant
- Analyze Stacktrace Dialog
- Artifacts to Deploy Dialog
- Bookmarks Dialog
- Breakpoints
- Build File Properties
- Choose Local Paths to Upload Dialog
- Code Duplication Analysis Settings
- Color Picker
- Composer Settings Dialog
- Configure Library Dialog
- Confirm Drop dialog
- Convert Project Dialog
- Create Jar from Modules Dialog
- Create Library Dialog
- Create New Constructor
- Create New Method
- Create New PHPUnit Test
- Create Run/Debug Configuration for Gradle Tasks
- Create Table and Modify Table dialogs
- Create Test
- CSV Formats Dialog
- Differences viewers
- Docker Registry Dialog
- Downloading Options Dialog
- Edit as Table: <file_Name> Format Dialog
- Edit Macros Dialog
- Edit Library Dialog
- Edit Project Path Mappings Dialog
- Evaluate Expression
- Export to Eclipse Dialog
- Export to HTML
- File Cache Conflict
- Find and Replace in Path
- Find Usages Dialogs
- Generate Ant Build
- Generate Equals() and HashCode() Wizard
- Generate Groovy Documentation Dialog
- Generate JavaDoc Dialog
- Generate ToString() Dialog
- Generate ToString() Settings Dialog
- Getter and Setter Templates Dialog
- Generate Getter Dialog
- Generate Setter Dialog
- Gradle Project Data to Import Dialog
- I18nize Hard-Coded String
- Incoming Connection Dialog
- IntelliJ IDEA License Activation Dialog
- JetBrains Decompiler Dialog
- Login to IntelliJ Configuration Server Dialog
- Manage Project Templates Dialog
- Map External Resource Dialog
- Non-Project Files Protection Dialog
- New Action Dialog
- Open Task Dialog
- Optimize Imports Dialog
- Override Server Path Mappings Dialog
- Play Configuration Dialog
- Print
- Productivity Guide
- PSI Viewer

- Pull Image Dialog
- Push Image Dialog
- Recent Changes Dialog
- Refactoring Dialogs
- Reformat File Dialog
- Reformat Code on Directory Dialog
- Rules Alias Definitions Dialog
- Register New File Type Association Dialog
- Resource Bundle Editor
- Run/Debug Configurations
- Save Project as Template Dialog
- Select Path Dialog
- Setup Library Dialog
- Show History for File / Selection Dialog
- Show History for Folder Dialog
- Specify Dependency Analysis Scope Dialog
- Specify Code Duplication Analysis Scope
- Specify Code Cleanup Scope Dialog
- Specify Inspection Scope Dialog
- Structural Search and Replace Dialogs
- Type Migration Dialog
- Web Server Debug Validation Dialog

Add Composer Dependency Dialog

Tools | Composer | Add Dependency

In this dialog box, appoint the packages to be added to the current project.

Item	Description
Available packages	From this list, select the package that you need in your project. The list shows all the available packages, however, the packages that have been already installed are marked with a tick. Use the search field, if necessary: start typing the search string, as you type, the list dynamically reduces to show the packages that match the entered pattern. The Description read-only text box briefly explains the functionality of the selected package.
Version to install	From this drop-down list, select the package version. The contents of the list depend on the specific package.
Install	When you click this button, IntelliJ IDEA starts downloading the appointed package. When the process is completed, IntelliJ IDEA creates a new folder under the Vendor node stores the downloaded package in it, and adds the package to the list in the `require` section of `Composer.json`.
Settings	In this area, specify the advanced installation options: – Path to PHP executable: in this field, specify the location of the PHP engine installation folder. Choose the PHP installation from the list, or type the path manually, or click the Browse button ... and choose the desired location in the dialog box that opens. – Command line parameters: in this text box, type the additional command line parameters to adjust the add dependency procedure. For example, to have a dependency added to the `require-dev` section instead of the default `require` section type `-dev`. For more information about Composer command line options during installation, see https://getcomposer.org/doc/03-cli.md .

Add Frameworks Support dialog

Select the technologies, frameworks and languages to be supported. If necessary, also select an application server that you want to use to deploy and run your application.

As a result, you'll be able to download and configure the necessary framework components (e.g. implementation libraries). You will also have various descriptors, configuration files and sample code generated automatically, and the module dependencies properly configured.

In IntelliJ IDEA terms, the necessary facets will be created along with an artifact specification and a run/debug configuration , if appropriate.

Note that the set of frameworks and technologies you can select from depends on which plugins are currently enabled in IntelliJ IDEA. For example, Ruby and Python are available only if the corresponding plugins are installed and enabled. For more information, see Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Also note that the list may vary depending on which frameworks and technologies are already enabled in your module, and on the module type .

- Web Application
- Struts
- Struts 2
- WebServices
- JSF
- Primefaces, Richfaces, Openfaces, or Icefaces
- Google App Engine
- Groovy
- Hibernate
- JavaEE Persistence
- JBoss Drools
- OSGi
- SQL Support
- Thymeleaf
- WebServices Client
- Batch Applications
- CDI: Contexts and Dependency Injection
- DM Server
- EJB: Enterprise JavaBeans
- Google Web Toolkit
- JavaEE Application
- RESTful WebServices
- Tapestry
- Spring
- Spring MVC, Spring Batch, or other Spring framework
- Application Server

Web Application

Select the check box to enable generic Web application development support. See also, Enabling Web Application Support .

Item	Description
Version	Select the version of the Servlet specification to be supported.
Create web.xml	For version 3.0 or later: select this check box to create the deployment descriptor file `web.xml` . (For earlier versions, this file is always created.)

Struts

Select the check box to enable Apache Struts 1.x support. See also, Preparing to Use Struts .

Item	Description
Version	Select the Struts version to be supported. If you also choose to download the library files that implement Struts (the Download option), the selected version will define which files you will be able to download.
Libraries	You'll need a library that implements Struts. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement the selected Struts version. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Struts 2

Select the check box to enable Apache Struts 2 support. See also, Preparing to Use Struts 2 .

You'll need a library that implements Struts 2. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the library files that implement Struts 2. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
- Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

WebServices

Select the check box to enable Web Services development support. See also, Preparing to Develop a Web Service .

Item	Description
Generate sample server code	Select this check box to have a sample `HelloWorld` class created in your source folder (e.g. `src`).
Configure	Click this link to specify the settings for WS engines that you want to use. (The Web Services dialog will open.)

JSF

Select the check box to enable JavaServer Faces (JSF) support. See also, Preparing for JSF Application Development .

Item	Description
Version	Select the JSF version to be supported.
Libraries	You'll need a library that implements JSF. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement JSF. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Primefaces, Richfaces, Openfaces, or Icefaces

Select the check box to be able to use the corresponding JSF component library (PrimeFaces , RichFaces , OpenFaces , or ICEfaces). See also, Preparing for JSF Application Development .

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the corresponding JSF component library files. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Google App Engine

Select the check box to be able to use Google App Engine . See also, Creating Google App Engine Project .

Item	Description
Google App Engine SDK	Specify the path to the Google App Engine SDK for Java installation directory. You can click ... and select the corresponding directory in the dialog that opens .
Persistence	If necessary, select the App Engine Datastore implementation to be supported (JDO or JPA).
Download	If the path to Google App Engine SDK is not specified, you can click this link to open the Google App Engine Downloads page . (This page lets you download the latest version of Google App Engine SDK for Java.)

Groovy

Select the check box to enable Groovy support. See also, Creating Modules with Groovy and Groovy-Based Frameworks Support .

Select an existing Groovy library or create a new library for Groovy:

- Use library. Select the Groovy library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA)

– Use library. Select the Groovy library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
– Create. If Groovy is already installed on your computer, you can create a library for Groovy and use that new library. To do that, click Create and select the Groovy installation directory in the dialog that opens .

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

Hibernate

Select the check box to enable Hibernate support. See also, Enabling Hibernate Support .

Item	Description
Create default hibernate configuration and main class	Select this check box to have a Hibernate configuration file `hibernate.cfg.xml` and a class with the `main()` method created.
Import database schema	Select this check box to have a database schema imported automatically.
Libraries	You'll need a library that implements Hibernate. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement Hibernate. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

JavaEE Persistence

Select the check box to enable Java Persistence API (JPA) support. See also, Enabling JPA Support .

Item	Description
persistence.xml version	Select the version of the `persistence.xml` file that you want to use. If you also choose to download the library files that implement JPA (the Download option), the selected version will define which files you will be able to download.
Import database schema	Select this check box to have a database schema imported automatically. Optionally, select the JPA implementation-specific database scheme to be imported from the list above the check box.
Libraries	You'll need a library that implements JPA. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement the selected JPA version. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

JBoss Drools

Select the check box to enable JBoss Drools support.

You'll need a library that implements Drools. You can choose to use an existing library, create and use a new one, or download the library files if they are not yet available on your computer.

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
– Download. Select this option to download the library files that implement Drools. (The downloaded files will be arranged in a library .)
Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

OSGi

Select the check box to enable OSGi support.

You'll need a library that implements OSGi. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

– Download. Select this option to download the library files that implement OSGi. (The downloaded files will be arranged in a library .)
Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

– Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

SQL Support

Select the check box to enable SQL support.

Item	Description
Default Dialect	Select the SQL dialect to be used by default for the module. Select Project Default to use the default project SQL dialect.

Thymeleaf

Select the check box to enable Thymeleaf support.

You'll need a library that implements Thymeleaf. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

– Download. Select this option to download the library files that implement Thymeleaf. (The downloaded files will be arranged in a library .)
Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

– Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

WebServices Client

Select the check box to enable Web Services client development support. See also, Enabling Web Service Client Development Support .

Item	Description
Generate sample client code	Select this check box to have sample client code generated in your source folder (e.g. `src`). To generate the code, IntelliJ IDEA will ask you to specify the corresponding WSDL file.
Configure	Click this link to specify the settings for WS engines that you want to use. (The Web Services dialog will open.)

Batch Applications

Select the check box to enable Batch Applications development support.

Item	Description
Create batch.xml	Select the check box to create a `META-INF\batch.xml` mappings file (one with the `<batch-artifacts>` root element).
Create Sample Job Xml	Select the check box to create a sample job XML file (`META-INF\batch-jobs\job.xml`).
Libraries	You'll need a library that implements the batch framework. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the Ctrl key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the files that implement the batch framework. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

CDI: Contexts and Dependency Injection

Select the check box to enable Contexts and Dependency Injection (CDI) support.

You'll need a library that implements CDI. You can choose to use an existing library, create and use a new one, or download the library files if they are not yet available on your computer.

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

– Download. Select this option to download the library files that implement CDI. (The downloaded files will be arranged in a library .)
Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

DM Server

Select the check box to be able to use SpringSource dm Server and develop dm Server-targeted applications.

Item	Description
Server	Select the server to be used from the list (if the corresponding server is already defined in IntelliJ IDEA). To define a server in IntelliJ IDEA, click Add and specify the server settings in the Spring dmServer dialog that opens.
Facet	Select the deployment artifact type that the module will implement (the Spring DM facet type in IntelliJ IDEA terms): – Bundle, an OSGi bundle. IntelliJ IDEA will create: – The `META-INF\MANIFEST.MF` file for the bundle. – A dm Bundle artifact configuration . – A project-level library for the dm Server API. This library will be added to the module dependencies . – A dm Server-oriented run/debug configuration . If necessary, specify additional options . – PAR or Plan, a dm Server PAR or plan. Specify the associated settings . – Configuration, a configuration artifact. Specify the artifact name . For more information on dm Server deployment artifacts, see "Deployment Architecture" and "Developing Applications" in SpringSource dm Server Programmer Guide .
Bundle options	
Spring DM Support	Select this check box to enable Spring support (to create a Spring facet in IntelliJ IDEA terms). As a result, IntelliJ IDEA will create the following files: – `META-INF\spring\module-context.xml` – `META-INF\spring\osgi-context.xml` At this step you are not suggested to download the library files that implement Spring. However, you will be able to do that after the module has been created by using the corresponding quick fix in the Project Structure dialog. For more information, see Adding a Spring facet to an existing module.
Web module	Select this check box to enable generic Web application development support (to create a Web facet in IntelliJ IDEA terms). Specify the associated settings: – Version. Select the version of the Servlet specification to be supported. – Create web.xml. For version 3.0 or later: select this check box to create the deployment descriptor file `web.xml` . (For earlier versions, this file is always created.) As a result, IntelliJ IDEA will create `web\WEB-INF\web.xml` (for version 3.0 or later - if so specified).
PAR or Plan options	
Name	For a PAR, this is the application identifier (`Application-SymbolicName`), for a plan - the plan name (the `name` attribute of the `<plan>` element).
Version	The application or the plan version (`Application-Version` or the `version` attribute of the `<plan>` element).
Plan	Select this option to create a plan. IntelliJ IDEA will create: – A `.plan` XML file. – A dm Plan artifact specification.
Platform Archive (PAR)	Select this option to create a PAR. IntelliJ IDEA will create: – The `META-INF\MANIFEST.MF` file for the PAR. – A dm Platform Archive artifact specification.
Scoped	For a plan: select this check box to make the plan scoped (corresponds to `scoped="true"` within the `<plan>` element).
Atomic	For a plan: select this check box to make the plan atomic (corresponds to `atomic="true"` within the `<plan>` element).
Nested bundles	Use the controls in this area to manage other dm Server deployment artifacts within the PAR or plan. (In IntelliJ IDEA, these are represented by other modules within the same project if those modules have suitable dm Server facets. A PAR may include OSGi bundles and configuration artifacts; a plan - OSGi bundles, configuration artifacts, PARs and other plans). – Add. Use this button to add the artifacts to the list. Select the necessary artifacts (IntelliJ IDEA modules) in the dialog that opens. – Remove. Use this button to remove the selected artifacts from the list. – Up. For a plan: use this button to move the selected artifact one line up in the list. (The order of artifacts defines their deployment order.) – Down. For a plan: use this button to move the selected artifact one line down in the list. – Versions. For a plan: use this button to specify the version or the range of versions for the selected artifact (corresponds to the `version` attribute of the `<artifact>` element).
Configuration option	
Name	Specify the OSGi name of the artifact (at the deployment stage, corresponds to the name of the file).

EJB: Enterprise JavaBeans

Select the check box to enable Enterprise JavaBeans (EJB) support. See also, Enabling EJB Support .

Item	Description

Version	Select the EJB version to be supported. If you also choose to download the library files that implement EJB (the Download option), the selected version will define which files you will be able to download.
Libraries	You'll need a library that implements EJB. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the library files that implement the selected EJB version. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Google Web Toolkit

Select the check box to be able to use Google Web Toolkit (GWT). See also, Enabling GWT Support .

Item	Description
GWT SDK	Specify the path to the GWT SDK installation directory. You can click … and select the corresponding directory in the dialog that opens .
Create sample application	Select this check box to have a sample application created. Specify the package for the application classes in the field underneath.
Download GWT	If the path to GWT SDK is not specified: you can click this link to open the Google Web Toolkit Downloads page . (This page lets you download a GWT SDK.)

JavaEE Application

Select the check box to enable Java EE application development support. See also, Enabling Java EE Application Support .

Item	Description
Version	The Java EE version.

RESTful WebServices

Select the check box to enable RESTful Web Services (client and server) development support. See also, RESTful WebServices .

Item	Description
Generate server code	Select this check box to have a sample `HelloWorld` server class created in your source folder (e.g. `src`).
Generate client code	Select this check box to have a sample `HelloWorldClient` class created in your source folder (e.g. `src`).
Libraries	You'll need a library that implements the JAX-RS API. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time. – Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA). Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.) Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .) – Download. Select this option to download the files that implement the JAX-RS API. (The downloaded files will be arranged in a library .) Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.) – Set up library later. Select this option to postpone setting up the library until a later time. Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Tapestry

Select the check box to enable Apache Tapestry support. See also, Enabling Tapestry Support .

You'll need a library that implements Tapestry. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

– Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)

– Download. Select this option to download the library files that implement Tapestry. (The downloaded files will be arranged in a library .)
Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

– Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Spring

Select the check box to enable Spring support. See also, Enabling Spring Support .

You'll need a library that implements Spring. You can choose to use an existing library, create and use a new one, download the library files if they are not yet available on your computer, or postpone setting up the library until a later time.

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the library files that implement Spring. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)
- Set up library later. Select this option to postpone setting up the library until a later time.

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Spring MVC, Spring Batch, or other Spring framework

Select a check box to add support for a particular Spring framework (e.g. Spring MVC , Spring Batch , etc.). See also, Enabling Spring Support .

You'll need a library that implements the selected framework. You can choose to use an existing library, create and use a new one, or download the library files if they are not yet available on your computer.

- Use library. Select the library to be used from the list (if the corresponding library is already defined in IntelliJ IDEA).
  Create. If the corresponding library files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)

  Optionally, click Configure to edit the selected library. (For an existing library the Edit Library Dialog will open, for the library that you have just created - the Create Library Dialog .)
- Download. Select this option to download the library files that implement the selected Spring framework. (The downloaded files will be arranged in a library .)
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Configure. Click this button to edit the settings for the library selected next to Use library or the one that is about to be downloaded.

Application Server

Select the check box and specify the application server that you are going to use to deploy and run your application. As a result, IntelliJ IDEA will create a run/debug configuration for the specified server.

Item	Description
Application Server	Select a server which IntelliJ IDEA is already aware of, or specify another "new" server. To specify a new server, select New <Server Name> from the list. Then, in the field or fields that appear underneath, specify the server settings: – For a server installed locally, specify the path to the server installation directory in the <Server Name> Home field. (Click … to select the directory in the corresponding dialog .) – For a hosted server (Cloud Foundry or CloudBees), specify your user account details.

Add New Field or Constant

UML Class diagram - Context menu of a node element | New | Field

`Alt+Insert`

Item	Description
Visibility	From this drop-down list, select the desired visibility modifier.
Type	Specify the type of the new field. Note that code completion works here. As you type, the suggestion list shrinks to show the matching options. Non-existent type is marked as a syntax error.
Name	Specify the name of the new field.
Initializer	Specify the initial value of the new field. Note that such value should correspond to the declared type. For example, if the type is String, the suggested initializer should be enclosed in quotes. Note that code completion works here. As you type, the suggestion list shrinks to show the matching options. Non-existent type is marked as a syntax error.
Static	Select this check box to add `static` to the field declaration.
Final	Select this check box to add `final` to the field declaration.
Preview	This read-only area shows the draft declaration as you fill in the dialog fields.

Analyze Stacktrace Dialog

Analyze | Analyze Stacktrace

Use this dialog box to reach a navigable console stack trace for external applications. From each message in this stack trace, you can navigate right to the source code that cased the reported problem.

Item	Description
Unscramble stacktrace	Select this check box to unscramble the external stack trace, if your source code is scrambled.
Unscrambler	Here you can select the unscrambler tool. IntelliJ IDEA ships with the Zelix Klass Master unscrambler plugin. You can develop your own plugin to unscramble stack trace of the code being processed with any other obfuscator.
Log file	Specify the location of the unscrambler log file.
Put a stack trace or a complete thread dump here	Paste here the external stack trace or thread dump.
Automatically detect and analyze thread dumps copied to the clipboard outside of IntelliJ IDEA.	If this check box is selected, IntelliJ IDEA will monitor and analyze the contents of the clipboard. You can select this check box only once and your clipboard will be scanned every time you switch to IntelliJ IDEA. As soon as something looking like a stack trace gets copied to the clipboard, IntelliJ IDEA will show this stack trace in the corresponding tool window.
Show changed in last <this_many> days	Select this check box to specify the time period in which you want to check last changes in stack traces. The default time period is set to 31 days.
Normalize	If the stack trace text is corrupted (lines are cut or wrapped, or too long, etc.) after processing with some software (for example, bug tracker or mail client), click this button to restore the normal stack trace structure.

Artifacts to Deploy dialog

Move the artifacts that you want to deploy to a server to the pane under Chosen.

Item	Description
Add	Move the artifacts selected in the Available pane to the Chosen pane.
Remove	Move the artifacts selected in the Chosen pane to the Available pane.
Add all	Move all the artifacts to the Chosen pane.
Remove all	Remove all the artifacts from the Chosen pane.

Bookmarks Dialog

Navigate | Bookmarks | Show Bookmarks

`Shift+F11`

Use this dialog to navigate between bookmarks, and manage the collection of anonymous bookmarks and bookmarks with mnemonics in a project.

Toolbar options

Item	Shortcut	Description
	`Ctrl+Enter`	Click to add/edit description for the selected bookmark.
	`Alt+Delete`	Click to delete the selected bookmark.
	`Alt+Down` `Alt+Up`	Use these buttons to reorder bookmarks.

The left pane of the Bookmarks dialog displays the list of bookmarks created with the brief description, if any. The order of bookmarks in the collection defines the order in which Navigate | Bookmarks | Next/Previous Bookmark command visits the bookmarks. The right pane displays the preview of the file where the selected bookmark is toggled.

To close the dialog box, press `Escape`.

Breakpoints

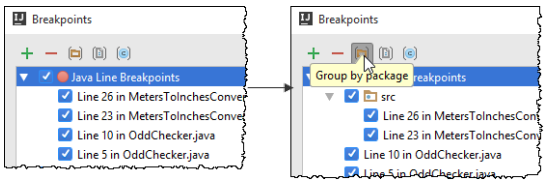
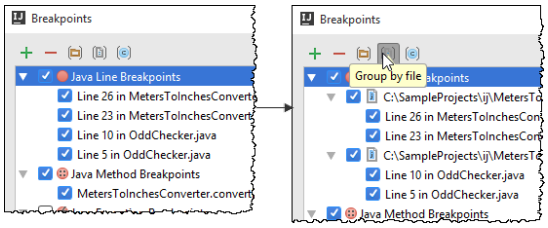
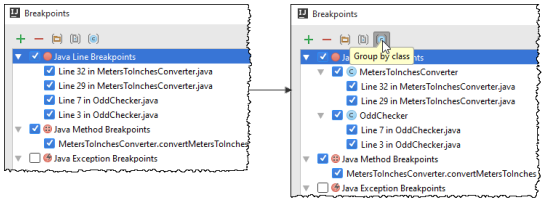
Run | View Breakpoints

`Ctrl+Shift+F8`

In this section:

- Toolbar
- Breakpoint options
- Context menu commands
- Speed search of a breakpoint

Toolbar

Item	Tooltip and shortcut	Description
+	Add Breakpoint `Alt+Insert`	Click to show the list of available breakpoint types . Select the desired type to create a new breakpoint.
−	Remove Breakpoint	Click this button to remove selected breakpoints.
	Group by Package	Click this button to display breakpoints under their respective packages, rather than under their types: 
	Group by File	Click this button to display breakpoints under their respective files, rather than under their types: 
	Group by Class	Click this button to display breakpoints under their respective classes, rather than under their types: 

Breakpoint options

The controls of this part of the Breakpoints dialog depend on the type of the selected breakpoint.

Option	Description	Types of breakpoints
Suspend	Select this check box to enable suspend policy for a breakpoint. For Line/ Method/ Exception breakpoints and Field Watchpoints select one of the radio buttons to specify the way the running of the program is paused when a breakpoint is reached. If you work with Flex or JavaScript breakpoints, you only need to specify whether you want to suspend program execution when the breakpoint is hit. If the check box is not selected, no threads are suspended.	Line/Exception/Field/Method

Suspend policy	Description
All	When a breakpoint is hit, all threads are suspended.
Thread	When a breakpoint is hit, the thread where the breakpoint is hit, is suspended.
Make default	Click this button if you want the suspend policy specified for the breakpoint in question to be used as the default one for the subsequently created breakpoints. This button only appears, when Thread option is selected.

- This feature is useful, for instance, to create a master breakpoint, which, being hit, enables lots of dependent ones. Another way to use is to obtain logging information or calculate some expression at a certain point (to be shown in the console) not interrupting the program execution.

- There are certain cases when IntelliJ IDEA will not stop at a breakpoint. Consider the following:
  1. Two breakpoints are set at the different methods of a class, and their suspend policy is set to All.
  2. When one of the breakpoints is hit, some step actions are performed.
  3. If at the time of stepping another thread hits the second breakpoint, IntelliJ IDEA will not stop there.

Condition	Select this check box and specify a condition for hitting a breakpoint in the text field. A condition is a Java Boolean expression (including a method returning `true` or `false`), for example, `str1.equals(str2)`. This expression should be valid at the line where the breakpoint is set, and is evaluated every time the breakpoint is reached. If the evaluation result is `true`, user-selected actions are performed. If evaluation result is `true`, user-selected actions are performed. If the result is `false`, the breakpoint does not produce any effect. If the Debugger cannot evaluate the expression, it displays the Condition evaluation error message. You can select whether you would like to stop at this breakpoint or ignore it. Conditions for field/method/exception breakpoints are calculated in the context for the given field/method/exception. To the right of the Condition field, there is the button (`Shift+Enter`) that opens the multiline editor. Thus it's possible to enter multiline expressions, for example: `if (myvar == expectedVariable) { System.out.println (myvar); anotherVariable = true; } return true;`	All types
Log message to console	Select this check box if you want a log message to be displayed in the console output when the breakpoint is hit.	All types
Evaluate and log	Select this check box if you wish to evaluate a certain expression at this breakpoint and to export result to the console output. To the right of this field, there is the button (`Shift+Enter`) that opens the multiline editor. Tip If the expression to be evaluated is incorrect when a particular breakpoint is reached, the console output displays an error message: `Unable to evaluate expression <your_expression>`	Line breakpoints
Remove once hit	Select this check box, if the you want the breakpoint to be deleted after hitting it.	All types
Disabled until selected breakpoint is hit	From the drop-down list, select the breakpoint in question. The option None corresponds to the always enabled breakpoint. Besides that, you can also choose the behavior of this breakpoint, when the selected one is hit: – Disable again – Leave enabled	All types
Filters		
Instance filters	An instance filter is used to limit breakpoint hits only with particular object instances using instance IDs. The instance ID value can be introduced manually or using the Instance Filters dialog box called by clicking the ellipsis button. Existing instance filters are indicated by the instance ID delimited with spaces.	Line/Exception/Field/Method

Class filters	Select this check box to have the breakpoint behave differently in relation to particular classes.
	Define the class filter to appoint the classes where you want the breakpoint to be hit and the classes where the breakpoint should not be triggered.
	
	Classes in a filter can be identified by their names or by means of class patterns.
	
	A class pattern is a string that may start or end with an asterisk (*). The asterisk in a pattern means any number (including zero) of any characters. The patterns are matched against fully qualified class names.
	
	The breakpoint behavior is different in relation to classes specified by their names or using class patterns.
	
	A filter specified through a class name points at the class itself as well as at all its subclasses (i.e. the classes directly or indirectly extending this one).
	
	A filter specified through a class pattern points at the classes whose fully qualified names match the pattern. The subclasses of such classes are selected only if their fully qualified names also match the specified pattern.
	
	You can define a class filter in two ways:
	– Click the Browse button and configure the filter in the Class Filters dialog box that opens.
	– Type the filter manually in the text box. Use the following syntax:
	  – Use spaces to separate class names and class patterns from each other.
	  – The classes to be excluded should have a minus in preposition.
	
	For example, the filter `package1.Class1 *s2 -package3.Class3` means that the corresponding breakpoint:
	– Should be triggered in the class `package1.Class1` and all its subclasses, and also in the classes whose fully qualified class names end in `s2`.
	– Should not be triggered in the class `package3.Class3` and all its subclasses.
		Line/Exception/Field/Method

Pass count	Specify the integer number, on which hit of the breakpoint it should be triggered. After the specified number of passes, the breakpoint is hit. This function is helpful for debugging loops or methods called several times. When the execution process comes to a breakpoint, where Pass count is set, the debugger reduces the count value by 1 and compares it to zero. If the comparison result is `true`, the breakpoint is hit. If it is `false`, no other actions are performed until the next breakpoint is reached. The Pass count condition can be satisfied only once. In other words, if you have a loop inside a method and the Pass count condition has been honored once, the breakpoint will not be hit the next time the said method is called. This option is only enabled, when Condition, Instance filters and Class filters options are disabled.	Line/Exception/Field/Method

Watch		
Field access	Stands for triggering breakpoint every time the field is accessed.	Field watchpoints
Field modification	This check box is selected when simple read attempts shouldn't cause the breakpoint to trigger.	Field watchpoints
Method entry	Stands for triggering breakpoint every time the method is entered.	Method breakpoints
Method exit	Stands for triggering breakpoint every time the method is exited.	Method breakpoints

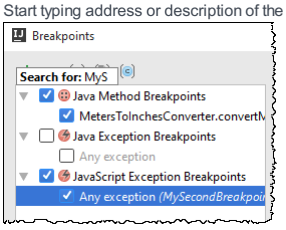
Context menu commands

Command	Description
Move to group	Point to this command to move the selected breakpoint to a new group, to one of the existing groups(`<group name>`), or out of a group (`<no group>`).
Edit description	Choose this command to enter or change description of a breakpoint.

Speed search of a breakpoint

To find a particular breakpoint

− Start typing address or description of the target breakpoint:



IntelliJ IDEA highlights the line with the matching address or description.

Class Filters Dialog

The dialog box opens when you select the Class Filters check box and click the Browse button ... in the Breakpoints dialog box.

Use this dialog box to configure class filters that determine in which classes a specific breakpoint will be hit and in which classes this breakpoint should not be triggered.

Item	Description
Class Filter	In this area, configure the set of classes where the selected breakpoint must be hit. Specify the classes either explicitly or using a class pattern. A class pattern is a string that may start or end with an asterisk (*). The asterisk in a pattern means any number (including zero) of any characters. The patterns are matched against fully qualified class names. The breakpoint behavior is different in relation to classes specified by their names or using class patterns. A filter specified through a class name points at the class itself as well as at all its subclasses (i.e. the classes directly or indirectly extending this one). A filter specified through a class pattern points at the classes whose fully qualified names match the pattern. The subclasses of such classes are selected only if their fully qualified names also match the specified pattern. – The list shows all the specified class names and class patterns. To have the breakpoint hit in the class or classes identified by a name or pattern, select the check box next to the corresponding item. – : click this button to open the Choose Class dialog box and specify a new class to be included in the filter. You can search for classes either by their names or using the project tree view. – : click this button to open the New Filter dialog box, where you can type the class pattern. – : click this button to have the selected item deleted from the list. Accordingly, the breakpoint will be no longer hit in the class or classes that match the removed name or pattern.
Class Exclusion Filter	In this area, configure the set of classes where hitting the selected breakpoint is suppressed. Specify the classes either explicitly or using a class pattern. – The list shows all the specified class names and class patterns. To suppress hitting the breakpoint in the class or classes identified by a name or pattern, select the check box next to the corresponding item. – : click this button to open the Choose Class dialog box and specify a new class to be included in the exclusion filter. You can search for classes either by their names or using the project tree view. – : click this button to open the New Filter dialog box, where you can type the class pattern. – : click this button to have the selected item deleted from the list. Accordingly, hitting the breakpoint in the class or classes that match the removed name or pattern will no longer be suppressed.

New Filter Dialog

The dialog box opens when you click in the Class Filter or Class Exclusion Filter area of the Class Filters dialog box.

In this dialog box, specify class patterns that identify the classes to be included in the currently configured class filter.

A class pattern points at the classes whose fully qualified names match the pattern. The subclasses of such classes are included in a filter only if their fully qualified names also match the specified pattern.

Item	Description
Enter the filter pattern	In this text box, specify the class pattern. A class pattern may start or end with an asterisk (*), which stands for any number (including zero) of any characters. The patterns are matched against fully qualified class names. Tip You can type a pattern manually or click the Browse button ... to open the Choose Class dialog box, locate the classes to be defined through the pattern, and compose the pattern based on this prompt. You can search for classes both by their names or in the project tree view.

Build File Properties

Ant Build tool window |

Ant Build tool window | Context menu of a build file | Properties

---

Use this dialog to configure the behavior of Ant build scripts.

Item	Description
Maximum Heap Size	Use this field to change the amount of memory allocated to the Ant build process. Increase this value if the Ant process runs out of memory.
Maximum Stack Size (MB)	Use this field to change the stack memory size. The default value is 2 MB.
Make build in background	If this option is enabled, build process runs in background. Otherwise, the modal progress dialog is displayed. **Warning!** Compilation requires significant processor resources and can result is serious slow-down of performance. When build process runs in the background, you can find IntelliJ IDEA not responding to your actions.
Close message view if no error occurred	If this option is enabled, Ant Messages window does not open for the successful builds.
Properties tab	Use this tab to specify the runtime properties that should be passed to the build script. These properties are equivalent to the ones defined after the `-D` option of the command line. A property is defined as a pair "name-value". The value can be hard-coded, or dynamically evaluated at runtime from a macro.
Add	Use this button to create a new entry in the list of runtime properties.
Remove	Use this button to delete the selected property from the list.
Execution tab	Use this tab to configure how IntelliJ IDEA will launch the build script.
Run with Ant	In this option group you can determine the Ant version to run the selected `build.xml` .
Use project default Ant	Click this radio button to run the default Ant version.
Use custom Ant	Click this radio button to run Ant version of your choice. From the combo box, you can select any Ant version that has been configured and registered with IntelliJ IDEA. Use the ellipsis button to modify Ant configuration.
Set Default	Click this button to configure the default Ant.
Ant command line	Use this field to supply command line arguments. You can include any arguments, except `-logger` . The arguments should be preceded with dashes and separated with spaces.
Run under JDK	Use this field to define JDK to use for running Ant target. By default, you can select the project or module JDK. Use the ellipsis button to configure JDK.
Additional Classpath tab	Use this tab to add libraries and directories that Ant loads at runtime.
Add	Click this button to add a directory of archive to the Classpath.
Add All In Directory	Click this button to add a directory with all its contents to the Classpath.
Move Up/Down	Use these buttons to change the order of classpath entries. The order of entries in the dialog defines the order in which Ant loads the resources.
Remove	Click this button to delete selected entry from the list.
Filters tab	Use this tab to configure which build targets show in the Ant Build tool window, when filtering is applied. The tab displays a list of build targets of the selected build file. The left column displays the names of the build targets, the right column contains optional description of the target. If a target is checked, it will show up in the Ant Build tool window after pressing ; otherwise such target will be hidden.

Choose Local Paths to Upload Dialog

The dialog box opens when you click the Upload to Web Server check box in the Run/Debug Configuration dialog box and click the Browse button … next to it.

Use the dialog box to configure a list of folders and files to be uploaded according to previously defined mappings and to edit these mappings, if necessary.

Item	Description
Add	Click this button to have a new row added to the list and specify the item to upload. Type the path manually or click the Browse button … to select the desired location in the Choose Path dialog box, that opens.
Remove	Click this button to remove the selected entry from the list.
Edit Mappings	Click this button to open the Deployment dialog box, where you can define mappings between local folders and the corresponding paths on the FTP/SFTP server.

Code Duplication Analysis Settings

Analyze | Locate Duplicates - OK

Use this dialog to define the sensitivity of search, and set limitation that will help you avoid reporting about every similar code construct. Your preferences are specified in a language-specific context.

ItemDescription

CSS

- Do not show duplicates containing less than <number> CSS properties : Set the size of duplicated language constructs that are shown in the results window.

CoffeeScript

- Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by IntelliJ IDEA.
  The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

ECMA Script level 4

- Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by IntelliJ IDEA.
  The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

Groovy

- Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by IntelliJ IDEA.
  The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

HTML

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).

Java

On this page, configure your preferences of search in Java constructs.

- Selecting each of the check boxes defines which should be anonymized.

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Visible from outside of the scope only: If this check box is selected, verify that the discarded subelement is valid outside the current construct. If the subelement is senseless, it cannot be discarded and should not be considered duplicated.
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by IntelliJ IDEA.
  The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

JavaScript

- Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by IntelliJ IDEA.
  The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

JSON

- Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by IntelliJ IDEA.
  The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

PHP

- Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by IntelliJ IDEA.
  The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

TypeScript

- Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by IntelliJ IDEA.
  The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

XHTML

- Do not show duplicates containing less than <number> tags : Set the size of duplicated language constructs that are shown in the results window.
- Anonymize values of tags and attributes

XML

- Do not show duplicates containing less than <number> tags : Set the size of duplicated language constructs that are shown in the results window.
- Anonymize values of tags and attributes

Color Picker

Settings | Editor | Colors and Fonts

Ctrl+Shift+A

Use this dialog to find the exact RGB, HSB and hex values of any color component.

Item	Description
	Click this eyedropper to navigate to the object and obtain its color RGB or HSB as well as its color hex values.
Color mixer	Use this area to select a color.
Stack of colors	This area displays colors that were last selected.
Brightness	Move the slider upwards to make to colors in the Color mixer area brighter; move the slider down to make colors darker.
Choose	Click this button to select a color from the Color Mixer area, or the Stack of colors.
Cancel	Click this button to omit saving the color information.

Composer Settings Dialog

Tools | Composer | Init Composer

In this dialog box, choose the instance of Composer to use in the current project among all the instances configured in IntelliJ IDEA. For more details, see Enabling and configuring the use of Composer Dependency Manager in IntelliJ IDEA .

Item	Description
Path to composer.phar	In this text box, specify the location of the `composer.phar` archive to use in the current project. Type the path manually or click the Browse button ... and choose the desired location in the dialog box that opens. IntelliJ IDEA parses the contents of the specified file for Composer commands.
Click here to download from getcomposer.org	Click this link to have IntelliJ IDEA download `composer.phar` from the official storage and specify the folder to store the archive in. This instance of Composer will be available in the current project only. To use it in the command line mode, configure it as a command line tool .

Configure Library Dialog

Use this dialog to edit the library name and to manage the library contents.

The set of the available controls depends on whether you are working with a Java or ActionScript/Flex library, or a JavaScript library.

– Controls for a Java or ActionScript/Flex library

– Controls for a JavaScript library

Controls for a Java or ActionScript/Flex library

Item	Description
Change Version	This button may be available for a library that implements a certain framework or technology (e.g. JSF, Spring) in cases when IntelliJ IDEA can make version-specific file replacements in the library. In such cases, when you click this button, the Downloading Options Dialog opens in which you can select the necessary library version, and also the files to be downloaded. As a result, the files in the library will be replaced with the downloaded files.
+ (Alt+Insert)	Use this icon or shortcut to add items (classes, sources, documentation, etc.) to the library. In the dialog that opens , select the necessary files and folders. For a Java library, these may be individual `.class` and `.java` files, directories and archives (`.jar` and `.zip`) containing such files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`). For an ActionScript/Flex library, these may be raw ActionScript 3 libraries, `.swc` , `.jar` and `.zip` files, the directories containing such files, and so on. IntelliJ IDEA will analyze the selected files and folders, and automatically assign their contents to the appropriate library categories (Classes, Sources, Documentation, Native Library Locations, etc.). When IntelliJ IDEA cannot guess the category (e.g. when you select an empty folder), a dialog will be shown, in which you will be able to specify the category yourself.
	To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.
	Click this icon to make certain library items "excluded" (see Excluded library items). In the dialog that opens, select the items that you want IntelliJ IDEA to ignore (folders, archives and folders within the archives), and click OK.
– (Delete)	When you click this icon or press Delete : – The selected "ordinary" library items are removed from the library. – The selected excluded items (see Excluded library items) become "ordinary" items, i.e. their excluded status is cancelled. The items themselves will stay in the library.

Controls for a JavaScript library

Item	Description
+ (Alt+Insert)	Use this icon or shortcut to add items to the library. Select one of the following options: – Attach Files or Directories. Select this option to add JavaScript files. In the dialog that opens, select the necessary files and folders. These may be individual JavaScript files and the directories containing such files. IntelliJ IDEA will analyze the selected files and folders, and automatically assign the JavaScript files to the appropriate categories. Minified files will be assigned to the Release category; ordinary (uncompressed) files will be assigned to the Debug category. When IntelliJ IDEA cannot guess the category (e.g. when you select an empty folder), a dialog will be shown, in which you will be able to specify the category (Release or Debug) yourself. – Attach Debug Version(s). Select this option to add a single uncompressed JavaScript file or a directory containing such files. Note that IntelliJ IDEA won't analyze the contents of the selected files. If you select a minified JavaScript file or a directory containing minified JavaScript files, the corresponding file or files will still be added to the library. – Attach Release Version(s). Select this option to add a single minified JavaScript file or a directory containing such files. Note that IntelliJ IDEA won't analyze the contents of the selected files. If you select an ordinary (uncompressed) JavaScript file or a directory with such files, the corresponding file or files will still be added to the library. – Specify Documentation URL. Select this option to make external online documentation available in IntelliJ IDEA. Specify the documentation URL in the dialog that opens. – Download Documentation. For jQuery: select this option to download and include jQuery documentation in the library.
	Click this icon to make certain library items "excluded" (see Excluded library items). In the dialog that opens, select the folders that you want IntelliJ IDEA to ignore, and click OK.
– (Delete)	When you click this icon or press Delete : – The selected "ordinary" library items are removed from the library. – The selected excluded items (see Excluded library items) become "ordinary" items, i.e. their excluded status is cancelled. The items themselves will stay in the library.

Confirm Drop dialog

From the Database tool window :

Edit | Drop, Drop from the context menu, or `Delete` .

Click OK to remove the selected item or items, or click to copy the SQL statement or statements into the database console .

Item	Description
SQL Preview	The statement or statements to be run to remove the selected item or items. If necessary, you can edit the statements right in this pane. The statements are executed when you click OK.
	Copy the statements into the corresponding database console and close the dialog.

Convert Project Dialog

This dialog box appears if you try to open a project created with earlier IntelliJ IDEA version.

File | Open Project

Item	Description
Convert to the new format now. You will not be able to open the project by earlier versions of IntelliJ IDEA	Select this option to convert the project to the new format. IntelliJ IDEA will back up old versions of project files to `projectFilesBackUp` folder within your project, so you could easily return to the old project version by replacing the project files with the old ones.
Convert later. Some features will be unavailable until you convert the project.	Select this option if you plan to work with this project using earlier IntelliJ IDEA versions. Some new features will be unavailable.

Create Jar from Modules Dialog

File | Project Structure | Artifacts - + - JAR - From modules with dependencies

Specify the settings for your Java archive (JAR).

Item	Description
Module	The module to be packaged.
Main class	The fully qualified name of your main application class, the one with a `main()` method.
JAR files from libraries	The way the JAR files from the module libraries are processed: – extract to the target JAR. The JAR contents are decompressed and then packaged together with the module output in a single JAR. – copy to the output directory and link via manifest. The JAR files are copied to the artifact output directory as is. The references to the JARs are added to the `Class-Path` header field of the `MANIFEST.MF` file that is packaged in the same JAR as the module output.
Directory for META-INF/MANIFEST.MF	The path to the directory in which `META-INF/MANIFEST.MF` is generated.
Include tests	Include the module's compiled test classes.

Create Library dialog

Specify the library name, level and contents.

Item	Description
Name	Use this field to edit the library name.
Level	Select the library level (global, project or module).
+ (Alt+Insert)	Use this icon or shortcut to add items (classes, sources, documentation, etc.) to the library. In the dialog that opens , select the necessary files and folders. For a Java library, these may be individual `.class` and `.java` files, directories and archives (`.jar` and `.zip`) containing such files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`). For an ActionScript/Flex library, these may be raw ActionScript 3 libraries, `.swc` , `.jar` and `.zip` files, the directories containing such files, and so on. IntelliJ IDEA will analyze the selected files and folders, and automatically assign their contents to the appropriate library categories (Classes, Sources, Documentation, Native Library Locations, etc.). When IntelliJ IDEA cannot guess the category (e.g. when you select an empty folder), a dialog will be shown, in which you will be able to specify the category yourself.
	To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.
	Click this icon to make certain library items "excluded" (see Excluded library items). In the dialog that opens, select the items that you want IntelliJ IDEA to ignore (folders, archives and folders within the archives), and click OK.
– (Delete)	When you click this icon or press Delete : – The selected "ordinary" library items are removed from the library. – The selected excluded items (see Excluded library items) become "ordinary" items, i.e. their excluded status is cancelled. The items themselves will stay in the library.

Create New Constructor

UML Class diagram - Context menu of a node element | New | Constructor

`Alt+Insert`

Item	Description
Parameters	
Type	Specify the parameter type. Note that code completion works here. As you type, the suggestion list shrinks to show the matching options. Non-existent type is marked as a syntax error.
Name	Specify the parameter name.
Add/Remove	Use these buttons to add/remove parameters.
Move Up/Down	Use these buttons to reorder parameters by moving them up or down in the parameter list.
Visibility	Select one of the options to change visibility scope of the constructor.
Exceptions	
Add/Remove	Use these buttons to add/remove exceptions. For each new exception, specify its type. Note that code completion works here. As you type, the suggestion list shrinks to show the matching options. Non-existent type is marked as a syntax error.
Move Up/Down	Use these buttons to reorder exceptions by moving them up or down in the list.
Signature Preview	This read-only area shows the draft constructor signature as you fill in the dialog fields.

Create New Method

UML Class diagram - Context menu of a node element | New | Constructor

`Alt+Insert`

Item	Description
Name	Use this field to modify the method name.
Return type	Use this field to modify the method return type. Code completion (`Ctrl+Space`) is available in this field, in the Type column of the Parameters area, and in the exception type list in the Exceptions area.
Parameters	
Type	Specify the parameter type. Note that code completion works here. As you type, the suggestion list shrinks to show the matching options. Non-existent type is marked as a syntax error.
Name	Specify the parameter name.
Add/Remove	Use these buttons to add/remove parameters.
Move Up/Down	Use these buttons to reorder parameters by moving them up or down in the parameter list.
Visibility	Select one of the options to change visibility scope of the method.
Abstract	If this check box is selected, the `abstract` modifier is added to the method signature.
Exceptions	
Add/Remove	Use these buttons to add/remove exceptions thrown by the method. For each new exception, specify its type.
Move Up/Down	Use these buttons to reorder exceptions by moving them up or down in the list.
Signature Preview	This read-only area shows the draft method signature as you fill in the dialog fields.

Create New PHPUnit Test

File | New | PHPUnit Test

Project tool window | Context menu of a class | New | PHPUnit Test

Use this dialog to configure generation of PHPUnit test class stubs.

The dialog box is available when the PHPUnit tool is installed on your machine and enabled in IntelliJ IDEA. For more information, see http://www.phpunit.de/manual/current/en/installation.html and Enabling PHPUnit Support .

Item	Description
Class To Test	In this area, specify the production class to generate a test class for. Type the fully qualified name of the class in the Fully Qualified Name text box. The specified name will be used to propose the Test Class Name. To use completion, press Control+Space and choose the relevant production class from the pop-up list.
Test Class	– Name: in this text box, specify the name of the test class to generate. IntelliJ IDEA automatically composes the name from the production class name as follows: `<production class>Test.php` . – Namespace: in this text box, specify the namespace the test class will belong to. IntelliJ IDEA completes the namespace automatically based on the value of the Directory text box. When the Directory text box is changed, the value of the Namespace text box is changed accordingly. To use completion, press Control+Space and choose the relevant namespace from the pop-up list. – Directory: in this text box, specify the directory where the file with the test class will be stored. By default, it is the directory where the production class is stored. To specify another folder, click the Browse button and choose the relevant folder in the dialog box that opens. To use completion, press Control+Space and choose the relevant folder from the pop-up list. – File Name: in this text box, specify the name of the file that will be created for the test class definition. By default, the name is the same as the test class name. However, if several production classes are defined in one single file, for each generated test class IntelliJ IDEA will create a separate file.

Create Run/Debug Configuration for Gradle Tasks

This dialog lets you create a run/debug configuration for the selected Gradle task. It appears when you right-click the task in the Gradle tool window and select Create [task-name].

Item	Description
Name	This field shows the name of your Gradle project and the Gradle task that you have selected in the Gradle tool window for the configuration.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Gradle Project	Use this field to specify the location of your Gradle project. You can either enter it manually or click the Browse button and point to the desired location in the dialog that opens . You can also click button to select an available Gradle module from the list of registered Gradle modules in your existing IntelliJ IDEA project. The list has a tree structure that might be useful if you have a Gradle multi-module project.
Tasks	This field shows the selected Gradle task. You can add other tasks to the selected one. Use spaces to separate one task from another.
VM Options	Use this field to specify VM options for your Gradle project. If you need more room to type, click next to the field to access the VM options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Script Parameters	Use this field as the Gradle command line to specify options for Gradle tasks. For the information on the syntax, see Gradle command line options . Please note that some of the Gradle commands options are not supported in IDE. In this case you will receive an error message indicating the problem.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .

- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files .

	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Create Table and Modify Table dialogs

– Upper part: GUI
– SQL Script
– Options

Upper part: GUI

The upper part of the dialog provides a GUI for defining the table structure, and associated constraints and indexes.

SQL Script

The pane under SQL Script shows the statement or statements to be run to achieve the result you have specified using the GUI.

You can use this pane just as a preview. You can also edit the statement or statements right there.

Options

Item	Description
Execute in database	Change data structure in your database by running the statement or statements. Execute. Execute the statements right away. If associated changes in database consoles are possible, the corresponding preview is shown. See Previewing changes. Preview. Preview potential associated changes prior to executing the statements.
Replace existing DDL	Replace the definition (usually, a `CREATE` statement) in the corresponding database console or SQL file with the statements shown under SQL Script. Replace. Perform the replacement right away. If associated changes are possible, the corresponding preview is shown. Preview. Preview potential associated changes prior to performing the replacement.
Open in editor	Copy the statements into the corresponding database console or SQL file. – Modify existing objects. In the generated set of statements, `ALTER` statements are preferred to `CREATE` statements. – Create modified objects. The changes you have made using the GUI are translated into a minimal set of `CREATE` and other statements. – Create all objects. The generated set always includes the `CREATE TABLE` statement, as if the whole table were created anew. To Editor. Switch to the corresponding editor tab right away. If associated changes are possible, the corresponding preview is shown prior to copying the statements. Preview. Preview potential associated changes.

Create Test

`Alt+Enter` - Create Test

`Ctrl+Shift+T` - Create New Test

Item	Description
Testing library	Use this drop-down list to select the testing framework to be used.
Fix	This button is available, when a library for the selected testing framework is missing. Click this button to add the jar archive of the selected testing framework to the list of module dependencies and libraries.
Class name	Enter the name of the test class to be generated, or accept default.
Superclass	For JUnit3, the superclass `junit.framework.TestCase` is suggested automatically. For the other supported frameworks, this field is blank.
Destination package	Specify the name of the package where the generated test class will be stored. You can select the desired package from the recent history drop-down list, or type it directly in the text field. So doing, if a package with the specified name does not exist, if will be created automatically. Clicking ... enables you to choose one of the existing packages within your project.

Generate
 	
`setUp()/@Before`
 	
`tearDown()/@After`
	Select these check boxes to include stub methods for test fixtures and annotations into the generated test class.

Show inherited methods	Select this option to show all methods, including the inherited ones.
Generate test methods for	In the table, select the check boxes next to the methods you want to generate test for.

CSV Formats dialog

To access this dialog:

- From any of the table views:

  Right-click the table and select Data Extractor | Configure CSV Formats.

- From the Database tool window :

  Right-click the table or view of interest, select Dump Data to File | Configure CSV Formats.

This dialog lets you specify the settings for converting table data into delimiter-separated values formats (e.g. CSV, TSV) and vice versa.

When working on the conversion settings, use the preview in the right-hand part of the dialog.

Item	Description
Formats	The list of the available delimiter-separated values formats is shown. Each format is a named set of corresponding conversion settings. Select the format whose settings you want to view or edit. Use +, −, ↑ and ↓ to create, delete and reorder the formats; to create a copy of the selected format.
Value separator	Select or type the character for separating individual values.
Row separator	Select or type the character for separating rows.
Null value text	The text to be used as a value if a cell contains `null` (an unknown value).
Add row prefix/suffix	Row prefix and suffix are character sequences which in addition to the row separator indicate the beginning and end of a row. If necessary, click the link and specify the row prefix and suffix in the fields that appear.
Quotation	Each line in the area under Quotation is a quotation pattern (see Quote values). A quotation pattern includes: − The left quotation character, the one inserted before a value. − The right quotation character, the one inserted after a value; usually, the same as the left quotation character. − An escape method or character for the cases when the quotation character is part of a value. E.g. Escape: duplicate means that if a quotation character occurs within a value, it is doubled. (You can specify your own escape character instead.) If there is more than one pattern, the first of the patterns is used. Use +, −, ↑ and ↓ to create, delete and reorder the patterns. To start editing an existing pattern, just click the pattern of interest.
Quote values	Specify in which cases the values should be quoted (i.e. enclosed within quotation characters). − When needed. A value is quoted only if it contains the value and/or the row separator. − Always. Any value is quoted in its text representation.
Trim whitespaces	If this check box is not selected, the Unicode whitespace characters that precede and follow the value separators are treated as parts of the corresponding values. If this check box is selected, the corresponding whitespace characters are ignored or removed.
First row is header	If this check box is selected, the first row is treated as containing column names. The settings that appear under Header Format have the same meanings as the ones above but are applied to the first row.
First column is header	If this check box is selected, the first column is treated as containing row names.

Differences viewers

- Differences viewer for files
- Differences viewer for folders
- Differences viewer for table structures
- Differences viewer for tables

Differences viewer for files

Project tool window | context menu of a file | Compare File with Editor

Project tool window | context menu of a file | Compare Files

Version Control tool window | Repository and Incoming tabs |

Version Control tool window | context menu of a folder or file | Show Diff

In this section:

– Basics
– Diff & Merge viewer
– Keyboard shortcuts

Basics

This dialog is displayed every time you compare two files, or compare two versions of a file (local changes or changes between local files and their revisions in a remote repository). You can compare files of any types, including binaries and `.jar` files.

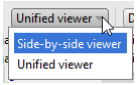
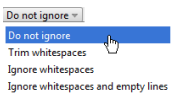
Note that you can open the differences viewer without running IntelliJ IDEA. To do this, execute the following command:

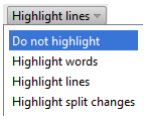
```
<path to IntelliJ
                    IDEA
                    executable file> diff <path_1>
                    <path_2>
```

The differences viewer provides a powerful editor that enables code completion, live templates, etc.

where `path_1` and `path_2` are paths to the files you want to compare.

Diff & Merge viewer

Item	Tooltip and Shortcut	Description
	Previous Difference / Next Difference `Shift+F7` `F7`	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, IntelliJ IDEA suggests to click the arrow buttons `F7` / `Shift+F7` once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings . This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.
	Compare Previous/Next File `Alt+Left` `Alt+Right`	Click these buttons to compare the local copy of the previous/next file with its update from the server. Tip These buttons are available only when the Differences Viewer is invoked from the Version Control tool window.
	Go To Changed File `Ctrl+N`	Click this button to display the path to the compared file. Tip This button is available only when the Differences Viewer is invoked from the Version Control tool window.
	Jump to Source `F4`	Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.
Viewer type		Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to – Edit, for example, type text. Note that one can change text only in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line. – Perform the Apply/Append/Revert actions.
Whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text. – Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default. – Trim whitespaces: `("\t", " ")` , if they appear in the end and in the beginning of a line. – If two lines differ in trailing whitespaces only, these lines are considered equal. – If two lines are different, such trailing whitespaces are not highlighted in the By word mode. – Ignore whitespaces: white spaces are not important, regardless of their location in the source code. – Ignore whitespaces and empty lines: the following entities are ignored: – all whitespaces (as in the 'Ignore whitespaces' option) – all added or removed lines consisting of whitespaces only – all changes consisting of splitting or joining lines without changes to non-whitespace parts. For example, changing `a b c` to `a \n b c` is not highlighted in this mode.
Highlighting mode		Select the way differences granularity is highlighted. The available options are:

		 – Highlight words: the modified words are highlighted – Highlight lines: the modified lines are highlighted – Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one. – Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	Collapse unchanged fragments	Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.
	Synchronize scrolling	Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.
	Editor settings	Click this button to reveal the drop-down list of available check commands. Select or clear these check commands to show or hide line numbers, indentation guides, white spaces, and soft wraps. Note that same commands are duplicated in the context menu of the differences viewer gutter. Right-click the gutter to show the menu of check commands.
	Include into commit `Alt+I`	This checkbox only appears if you invoke the Differences Viewer from the Commit Changes dialog with multiple changed files (all of which are deselected), and you explore the differences between them and hit the last difference in a file. Select this checkbox if you want to include the file you've reviewed into the commit.
	Move to Another Changelist `F6`	This button only appears if you invoke the Differences Viewer from the Commit Changes dialog with multiple changed files (all of which are deselected), and you explore the differences between them and hit the last difference in a file. Click this icon to move the file you've reviewed to another changelist.
	Show diff in external tool	Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff Tools settings page.
	Help `F1`	Click this button to show the corresponding help page.
	`Ctrl+Tab`	Use this keyboard shortcut to switch between the panes of the Differences viewer. The active pane has the cursor.
		Use these chevron buttons to apply differences between panes (in case of the side-by-side viewer) or between lines (in case of the unified viewer). The chevron buttons can change their behavior: – Click to apply changes. This behavior is the default one. – Press `Ctrl` to change to or and append changes.
Merge actions		
	Compare Left and Middle/Middle and Right/Left and Right Contents	Click these buttons to compare left/middle/right parts in a new window.
	Apply All Non-Conflicting Changes	Click this button to apply all non-conflicting changes. You can also make this behavior automatic, by selecting the check box Automatically apply non-conflicting changes in the Diff & Merge page of the Settings/Preferences dialog.
	Apply Non-Conflicting Changes from the Left/Right Side	Click these buttons to merge non-conflicting changes from the left/right parts of the dialog.

Keyboard shortcuts

Keyboard shortcut	Description
`Ctrl+Shift+D`	Use this keyboard shortcut to show the popup menu of the most commonly user diff commands.
`Ctrl+Tab`	Use this keyboard shortcut to switch between the left and the right panes.
`Ctrl+Shift+Tab`	Use this keyboard shortcut to select the position obtained by `Ctrl+Tab` in the opposite pane.
`Ctrl+Z` / `Ctrl+Shift+Z`	Use this keyboard shortcut to undo/redo a merge operation. Conflicts will be kept in sync with the text.

Differences viewer for folders

Project tool window | context menu of a folder | Compare Directory with

Project tool window | context menu of two selected folders | Compare Directories

Project tool window | context menu of a folder | Sync with Deployed to

Remote Host tool window | context menu of a folder | Sync with local

Tools | Deployment | Sync with local

Database tool window | context menu of two selected items | Compare

This window is displayed every time you explore differences between:

- Two local directories (local history for folders, recent changes, or version control).
- A remote folder and its local version .

In this dialog box, explore the detected differences and synchronize the compared items.

Tip You can also open the difference viewer without running IntelliJ IDEA. This is done through the following command: `<path to IntelliJ IDEA executable file> diff <path_1> <path_2>` where `path_1` and `path_2` are paths to the folders in question.

In this section:

- Toolbar
- List of items
- Differences pane
- Diff viewer
- Context menu

Toolbar

Item	Tooltip and shortcut	Description	Available for
	F7 Shift+F7	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, IntelliJ IDEA suggests to press F7 / Shift+F7 once more and compare other files. Tip This feature becomes available only when the Differences Viewer is invoked from the Version Control tool window.	Version control
	Jump to Source F4	Click this button to open file in the editor's active tab. The caret will be placed in the same position as in the Differences Viewer.	All
	Show diff in external tool	Click this button to invoke an external differences viewer, specified in the External Diff Tools page. This button only appears on the toolbar, when the check box Use external diff tool is selected in the External Diff Tools page.	All
	Help F1	Click this button to show reference page.	All
	Refresh F5	Click this button to refresh the contents of the differences viewer.	All
	Show new files on left side	Press this toggle button to have IntelliJ IDEA show the items, that are present in the first of the compared directories or database objects and are missing in the second one.	All
	Show difference	Press this toggle button to have IntelliJ IDEA show items that are present in both folders or database objects but differ in contents, or timestamp, or size.	All
	Show equals	Press this toggle button to have IntelliJ IDEA show the items that are present in both directories or objects and have the same contents, or timestamp, or size, depending on the parameter for comparison specified in the Compare by drop-down list.	All
	Show new files on right side	Press this toggle button to have IntelliJ IDEA show the items, that are present in the second of the compared directories or database objects and are missing in the first one.	All
Compare by		In this drop-down list, select the parameter to be used for comparison. The available options are: - Contents - Size - Time stamp	Local folders Local-remote folders
	Synchronize Selected	Click this button to have IntelliJ IDEA apply the specified action to the selected pair of items. Actions to be performed are shown in the * field.	All

	Enter Synchronize All	Click this button to have IntelliJ IDEA apply the specified action to all the pairs of items in the list. Actions to be performed are shown in the * field.	All
	Ctrl+Enter Hide excluded files	Click this button to suppress showing files excluded from synchronization .	Local-remote folders
Filter		Type the filtering string (for example, file or table name). Use * wildcard to replace any number of arbitrary characters. Note that filter applies on pressing Enter .	All
Path		These fields show the paths to the folders being compared. To change a directory, click the Browse button and specify another directory in the dialog that opens .	Local folders Local-remote folders
		These read-only fields show the names of the data sources or tables being compared.	Data sources

List of items

The list shows the items from the compared objects that meet the comparison criterion specified in the Compare by drop-down list and the filtering criteria specified through the toolbar buttons .

Item	Description	Available for
Name	These read-only fields show the names of files, data source tables, or table fields under the object specified in the Path or fields.	All
Size	These read-only fields show the sizes of files under the folders being compared.	Local folders Local-remote folders
Date	These read-only fields show the timestamps of files under the folders being compared.	Local folders Local-remote folders
*	The icon in this field indicates the action that will be applied to the pair of items in the current line upon clicking the Synchronize Selected or Synchronize All toolbar button. To change the currently selected action, click the icon.	All

Icon	Action
	Copy the item in the left side to the right side, possibly overwriting the contents of the corresponding target item, if it already exists.
	Copy the item in the right side to the left side, possibly overwriting the contents of the corresponding target item, if it already exists.
	The items are treated identical with regard to the selected criterion of comparison. No action will be performed by default.
	The items differ with regard to the selected criterion of comparison. No action will be performed by default. Explore the differences in the Differences Pane and change the intended action by clicking the icon.
	The item is present only in one of the folders and will be removed.

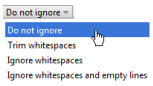
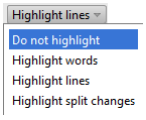
Differences pane

The differences pane is displayed only for files, data source tables, or table fields with the same names, which exist on both sides. For the files and DB objects that exist on one side only, the contents of the selected file/DB object is displayed.

If the files have read-only status, they are not editable in the differences pane.

Diff viewer

Item	Tooltip and Shortcut	Description
	Previous Difference / Next Difference Shift+F7 F7	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, IntelliJ IDEA suggests to click the arrow buttons F7 / Shift+F7 once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings . This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.
	Compare Previous/Next File Alt+Left Alt+Right	Click these buttons to compare the local copy of the previous/next file with its update from the server. Tip These buttons are available only when the Differences Viewer is invoked from the Version Control tool window.

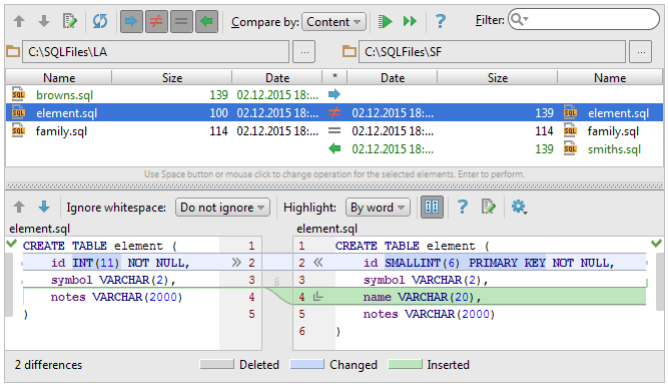
	Go To Changed File Ctrl+N	Click this button to display the path to the compared file. Tip This button is available only when the Differences Viewer is invoked from the Version Control tool window.
	Jump to Source F4	Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.
Viewer type		Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to – Edit, for example, type text. Note that one can change text only in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line. – Perform the Apply/Append/Revert actions.
Whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text. – Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default. – Trim whitespaces: `("\t", " ")` , if they appear in the end and in the beginning of a line. – If two lines differ in trailing whitespaces only, these lines are considered equal. – If two lines are different, such trailing whitespaces are not highlighted in the By word mode. – Ignore whitespaces: white spaces are not important, regardless of their location in the source code. – Ignore whitespaces and empty lines: the following entities are ignored: – all whitespaces (as in the 'Ignore whitespaces' option) – all added or removed lines consisting of whitespaces only – all changes consisting of splitting or joining lines without changes to non-whitespace parts. For example, changing `a b c` to `a \n b c` is not highlighted in this mode.
Highlighting mode		Select the way differences granularity is highlighted. The available options are: – Highlight words: the modified words are highlighted – Highlight lines: the modified lines are highlighted – Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one. – Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	Collapse unchanged fragments	Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.
	Synchronize scrolling	Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.
	Editor settings	Click this button to reveal the drop-down list of available check commands. Select or clear these check commands to show or hide line numbers, indentation guides, white spaces, and soft wraps. Note that same commands are duplicated in the context menu of the differences viewer gutter. Right-click the gutter to show the menu of check commands.
	Include into commit Alt+I	This checkbox only appears if you invoke the Differences Viewer from the Commit Changes dialog with multiple changed files (all of which are deselected), and you explore the differences between them and hit the last difference in a file. Select this checkbox if you want to include the file you've reviewed into the commit.
	Move to Another Changelist F6	This button only appears if you invoke the Differences Viewer from the Commit Changes dialog with multiple changed files (all of which are deselected), and you explore the differences between them and hit the last difference in a file. Click this icon to move the file you've reviewed to another changelist.
	Show diff in external tool	Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff Tools settings page.
	Help F1	Click this button to show the corresponding help page.
	Ctrl+Tab	Use this keyboard shortcut to switch between the panes of the Differences viewer. The active pane has the cursor.
» «		Use these chevron buttons to apply differences between panes (in case of the side-by-side viewer) or between lines (in case of the unified viewer). The chevron buttons can change their behavior: – Click » « to apply changes. This behavior is the default one. – Press Ctrl to change » « to ⇲ or ⇱ and append changes.

Context menu

This menu appears on right-clicking an entry in the list of items. The commands listed below define actions to be taken for the selected entry and set the action icons in the * column.

Item	Icon	Description
Set Copy to		This command sets the specified icons in the * column to copy a file missing from one of the directories. If a file exists on one side, it will be copied; if it doesn't

Right/Left	exist, then the file on the other side will not be deleted.
Set Delete	This command sets the specified icons in the * column to delete file.
Set Do Nothing	Choose this command to remove an action icon.
Set Default	Choose this command to set the default action for the entry.
Warn When Delete	Select this option to display a warning when trying to delete a file that is located only in one of the two directories during their merge.



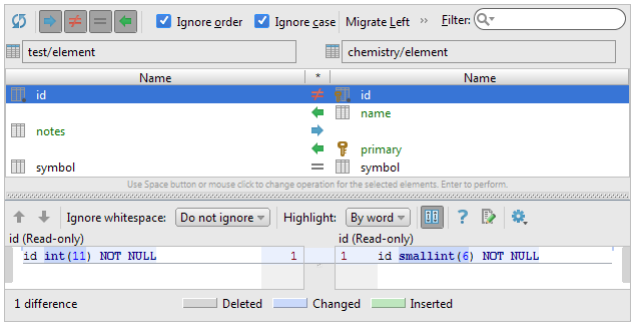
File comparison statuses and intended operations are shown in the column marked with an asterisk (*). To assign or change an operation, use the context menu associated with the corresponding cell. Alternatively, click the cell or press `Space` one or more times.

Applying operations:

- (`Enter`) applies the operations to selected files.
- (`Ctrl+Enter`) applies the operations to all the files.

Icon	Description
	The file exists only in the left-hand folder. The intended operation is to copy the file to the right-hand folder. If a file exists in both folders and you apply this operation, the file in the right-hand folder is overwritten.
	For the selected comparison criterion, the files are not identical. No operation is assumed. Study the file differences in the lower part of the view. You can choose to overwrite one of the files by assigning and applying the corresponding operation. You can as well modify the file contents. This may be done by typing or by using the following buttons and context menu commands: – or or Replace. Replace the fragment with the one from the other pane. – or or Insert. Insert the fragment into the other pane. – Remove. Remove the corresponding fragment. To undo the changes, use `Ctrl+Z`.
	For the selected comparison criterion, the files are identical. No operation is assumed.
	The file exists only in the right-hand folder. The intended operation is to copy the file to the left-hand folder. If a file exists in both folders and you apply this operation, the file in the left-hand folder is overwritten.
	Delete the file. The operation is not available for files that exist in both folders.

Differences viewer for table structures



Comparison statuses are shown in the column marked with an asterisk (*).

Icon	Description
	The column, constraint or index exists only in the left-hand table.
	The items exist in both tables but their definitions are different. You can study the differences in the lower part of the view.
	The items are identical.
	The item exists only in the right-hand table.

Generating ALTER TABLE statements

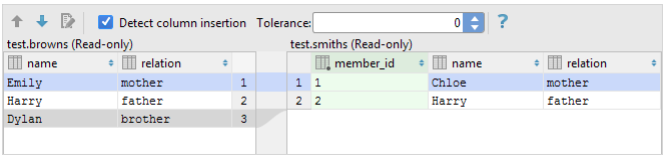
You can generate a set of `ALTER TABLE` statements for making data definitions in the left-hand and the right-hand parts identical. Use one of the following buttons in the upper part of the view:

- Migrate Left. The statements for the left-hand table or tables are generated.
- Migrate Right. This buttons does the same but for the right-hand table or tables.

The `ALTER TABLE` statements are generated for the items marked , and . If you don't want to generate the statements for some of those items, right click the * cell and select Set Do Nothing for all such items.

Differences viewer for tables

The primary purpose of the differences viewer for tables is to show the differences and similarities of data.



Docker Registry dialog

For this dialog to be available, the Docker integration plugin must be installed.

Specify your Docker image repository user account settings.

Item	Description
Name	The name for this set of settings.
Address	The image repository service URL, e.g. – registry.hub.docker.com for Docker Hub – quay.io for Quay
Username	The user name for your user account.
Password	Your password.
Email	The email address that you specified when creating your user account.
Server	The associated Docker Remote API configuration (used to connect to the service to check that your user account settings are correct).

Downloading Options dialog

Select which library files you want to download. Specify the destination directory and also other settings such as the library name and level .

Item	Description
Version	If available: select the library version you want to download. Note that your selection affects the set of files you can choose from (shown under Files to download).
Name	If necessary, edit the library name.
Level	If available: select the level which the library should be assigned to (global, project or module).
Files to download	Use the check boxes in this area to select the library files to be downloaded.
Download sources	If available: select this check box to download the source code for the library files.
Download javadocs	If available: select this check box to download Javadoc documentation for the library files.
Copy downloaded files to	Specify the path to the destination directory. To change the default path, click and select the directory in the dialog that opens .

Edit as Table: <file_name> Format dialog

Specify how your delimiter-separated values should be converted into table format.

When working on the conversion settings, use the table preview in the right-hand part of the dialog.

Item	Description
Formats	Select your file format and check the table preview. If you haven't achieved the desired result yet, adjust the settings. If you have changed the settings and want to save the changes, click this icon and select one of the following: – Save Changes. The settings are saved "under the same name", without creating a new format. (A format, in fact, is a named set of settings.) – Save As. The settings are saved "under a different name": a new format is created and you can specify the name for that new format.
Value separator	Select or type the character used for separating individual values.
Row separator	Select or type the character that should be treated as a row separator.
Null value text	The text to be used as a value if a cell contains `null` (an unknown value).
Add row prefix/suffix	Row prefix and suffix are character sequences which in addition to the row separator indicate the beginning and end of a row. If necessary, click the link and specify the row prefix and suffix in the fields that appear.
Quotation	Each line in the area under Quotation is a quotation pattern (see Quote values). A quotation pattern includes: – The left quotation character, the one inserted before a value. – The right quotation character, the one inserted after a value; usually, the same as the left quotation character. – An escape method or character for the cases when the quotation character is part of a value. E.g. Escape: duplicate means that if a quotation character occurs within a value, it is doubled. (You can specify your own escape character instead.) If there is more than one pattern, the first of the patterns is used. Use +, –, ↑ and ↓ to create, delete and reorder the patterns. To start editing an existing pattern, just click the pattern of interest.
Quote values	Specify in which cases the values should be quoted (i.e. enclosed within quotation characters). – When needed. A value is quoted only if it contains the value and/or the row separator. – Always. Any value is quoted in its text representation.
Trim whitespaces	If this check box is not selected, the Unicode whitespace characters that precede and follow the value separators are treated as parts of the corresponding values. If this check box is selected, the corresponding whitespace characters are ignored or removed.
First row is header	If this check box is selected, the first row is treated as containing column names. The settings that appear under Header Format have the same meanings as the ones above but are applied to the first row.
First column is header	If this check box is selected, the first column is treated as containing row names.

Edit Macros Dialog

Edit | Macros | Edit Macros

Use this dialog to change or delete the existing macros. Note that the dialog is not available, when there are no macros.

Item	Description
Existing macros list	The left-hand pane of the dialog shows the list of existing macros.
—	Click this button to delete a recorded macro.
	Click this button to change name of a recorded macro in the Rename Macro dialog.
Actions list	The right-hand pane shows the list of actions recorded for each macro. The list of actions for each macro is editable - you can remove unnecessary actions.
—	Click this button to delete an action from the macro.

Edit Library dialog

Edit the library name and contents.

Item	Description
Name	Use this field to edit the library name.
Change Version	This button may be available for a library that implements a certain framework or technology (e.g. JSF, Spring) in cases when IntelliJ IDEA can make version-specific file replacements in the library. In such cases, when you click this button, the Downloading Options Dialog opens in which you can select the necessary library version, and also the files to be downloaded. As a result, the files in the library will be replaced with the downloaded files.
+ (Alt+Insert)	Use this icon or shortcut to add items (classes, sources, documentation, etc.) to the library. In the dialog that opens , select the necessary files and folders. For a Java library, these may be individual `.class` and `.java` files, directories and archives (`.jar` and `.zip`) containing such files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`). For an ActionScript/Flex library, these may be raw ActionScript 3 libraries, `.swc` , `.jar` and `.zip` files, the directories containing such files, and so on. IntelliJ IDEA will analyze the selected files and folders, and automatically assign their contents to the appropriate library categories (Classes, Sources, Documentation, Native Library Locations, etc.). When IntelliJ IDEA cannot guess the category (e.g. when you select an empty folder), a dialog will be shown, in which you will be able to specify the category yourself.
	To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.
	Click this icon to make certain library items "excluded" (see Excluded library items). In the dialog that opens, select the items that you want IntelliJ IDEA to ignore (folders, archives and folders within the archives), and click OK.
– (Delete)	When you click this icon or press Delete : – The selected "ordinary" library items are removed from the library. – The selected excluded items (see Excluded library items) become "ordinary" items, i.e. their excluded status is cancelled. The items themselves will stay in the library.

Edit Project Path Mappings Dialog

The dialog box opens when you click … next to the Path Mappings field in various cases (for example, in the Run/Debug Configuration: Node JS dialog , or when you configure a remote interpreter).

If in the current run/debug configuration you use an interpreter accessible through SFTP connection or located on a Vagrant instance, the mappings are automatically retrieved from the corresponding deployment configuration or `Vagrantfile` and listed in the dialog box. The mappings are read-only.

- To add a custom mapping, click + and specify the path in the project and the corresponding path on the remote runtime environment in the Local Path and Remote Path fields respectively. Type the paths manually or click … and select the relevant files or folders in the dialog box that opens.
- To remove a custom mapping, select it in the list and click −.

Evaluate Expression

Run | Evaluate Expression

Editor Context Menu | Evaluate Expression

Alt+F8

Use this dialog box to calculate values of expressions or code fragments during the debugging session.

Item	Description
Expression	Use this field to edit the expression to be evaluated. If an expression is selected in the editor, this field displays selection. Tip This field is available in the Expression Mode.
Language	Choose Java or Groovy from the drop-down list, to enable recognition of the respective syntax of expressions. Tip This field is available only for Groovy scripts.
Statements to Evaluate	Type the group of statements to be evaluated. If a code fragment is selected in the editor, this field displays selection. Tip This field is available in the Code Fragment Mode.
Result	Here the results are displayed.
Evaluate	Click this button to evaluate the current expression or code fragment.
Close	Click this button to close the dialog box.
Code Fragment Mode	Click this button to toggle to the Code Fragment Mode.
Expression Mode	Click this button to toggle to the Expression Mode.

Export to Eclipse Dialog

Item	Description
Modules to export	This area shows the list of modules that have not been converted and switched to use the Eclipse format yet (the modules that have the IntelliJ IDEA module format `.iml`). Select the modules you want to export by selecting the corresponding check-boxes.
Switch selected modules to Eclipse-compatible format	If this check box is selected, the modules selected in the Modules to export area, will be switched to the Eclipse-compatible format.
Export non-module libraries	If this check box is selected, IntelliJ IDEA creates a User Library configuration file for Eclipse (`*.userlibraries`), containing definitions of all external libraries, used in the project. If this check box is not selected, the Path to resulting .userlibraries field is disabled.
Path to resulting .userlibraries	Specify the path to the generated `*.userlibraries` file. Note that this field is only available when the Export non-module libraries option is selected.

Export to HTML

File | Export to HTML

Use this dialog to save selected files in HTML format.

Item	Description
File <name>	Click this radio button to print the file currently selected in the Project view, or open in the editor.
Selected text	Click this radio button to print the text selected in the editor.
All files in the directory	Click this radio button to print all files in the current directory.
Include subdirectories	Select this check box to print the files in the subdirectories of the current directory.
Output directory	Specify fully qualified path to the directory, where the resulting HTML file will be stored.
Show line numbers	Select this check box to include line numbers in the resulting HTML file.
Generate hyperlinks to classes	Select this check box to replace class names with the hyperlinks to the respective classes.
Open generated HTML in browser	Select this check box to show HTML file in the default browser after export.

File Cache Conflict

An external process has changed a file, opened and unsaved in IntelliJ IDEA, which results in two conflicting versions of a file. Resolve this conflict using the following options.

Item	Description
Load FS Changes	Click this button to load the file version produced outside of IntelliJ IDEA, and overwrite your local changes.
Keep Memory Changes	Click this button to preserve the version produced in IntelliJ IDEA and stored in cache.
Show Difference	Click this button to invoke the differences viewer that shows the version in the file system to the left, and IntelliJ IDEA version to the right. You can merge differences as required and load the desired version to IntelliJ IDEA. By default, the cache version is saved.

Find and Replace in Path

Edit | Find | Find in Path or Replace in Path

`Ctrl+Shift+F` or `Ctrl+Shift+R`

In the Project tool window : Find in Path or Replace in Path from the context menu for a directory

---

Specify what you want to find and where. In the Replace in Path dialog, also specify the replacement text or pattern.

Use `Ctrl+Shift+F` and `Ctrl+Shift+R` to switch between the find and replace modes.

Search pattern and replacement text

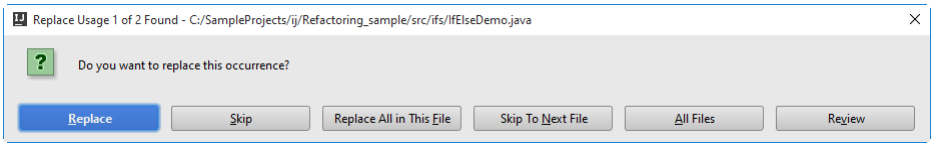
Item	Description
Text to find	In this field, specify the search pattern. Type the text manually or select one of the previously specified patterns from the drop-down list. – If you specify the search pattern through a regular expression, use the `$n` format in back references (to refer to a previously found and saved pattern). – This field can be left empty. If there is no text to find, but the File mask check box is selected, then search results include only the files matching the specified mask. Pressing `Alt+Down` in this field, will open a history drop-down list of your previous search entries.
Replace with	In this field, specify the replacement text. Type the text manually or select one of the previously specified substitutions from the drop-down list. – If you specify the replacement text through a regular expression, use the `$n` format in back references. – To use a backslash character `\` in a regular expression, escape the meaningful backslashes by inserting three extra backslashes in front: `\\\\` . This field is available only in the Replace in Path dialog.

Options

Use this tab to specify find and replace options.

Item	Description
General	In this area, specify additional search and replace parameters. The available options are: – Case sensitive - select this check box to have IntelliJ IDEA distinguish between upper and lowercase letters while searching. – Preserve case - if this check box is selected, IntelliJ IDEA retains the case of the first letter and the case of the initial string in general. For example, *MyTest* will be replaced with *Yourtest* if you specify *yourtest* as the replacement. This check box is disabled, if the Case sensitive or Regular expressions check box is selected. This field is available only in the Replace in Path dialog. – Whole words only (may be faster) - select this check box to have IntelliJ IDEA search for whole words only, that is, for character strings separated with spaces, tabs, punctuation, or special characters. This check box is disabled, if the Regular expressions check box is selected. – Regular expressions - select this check box if the specified search pattern should be treated as a regular expression . Tip Click the [Help] link next to the check box to view reference on regular expressions syntax. For more information on regular expressions and their syntax, refer to documentation for `java.util.regex` . – Context - use this drop-down list to confine the search to a certain context, for example: – anywhere - select this option to search everywhere. – In comments - select this option to confine search to comments, ignoring the other occurrences. – In string literals - select this option to confine search to string literals, ignoring the other occurrences – Except... - select one of the exception options to perform search avoiding comments, string literals or both.
Scope	In this area, specify the scope to apply the search to. The available options are: – Whole project - select this option to search through the entire project. – Directory - select this option to perform search within the specified directory. By default, the text area already contains the directory name where a file currently opened in the editor is located (if you call the dialog from the editor), or where a file selected in the tool window is located (if you call the dialog from the tool window), or the directory name selected in the tool window. Pressing the ellipsis button opens the Select Path dialog, where you can select the necessary directory. – Recursively - this check box is only available for the directory search. If selected, sets the search to be performed in the chosen directory and its subdirectories. – Custom - select this option to search in a scope . You can choose one of the scopes from the drop-down list, or click the ellipsis button, and define a new scope in the Scopes dialog .
File name filter	In this area, specify additional settings to narrow down the search scope. – File mask - select this check box to narrow down the search scope through file masks. In the drop-down list, select the desired mask or specify a new one using wildcards. Wildcards can include: – `*` to substitute a set of any characters, – `?` to substitute a single character, – `!` to exclude files. Mind that `!` should go first in a particular file name pattern, for example, `!*.gant` You can specify multiple file masks, delimited with commas (for example, `*.xml,a?c.sql,!*.html`). Note also, that negated pattern (for example, `!*.min.js`) has implicit inclusion pattern `*` . This allows avoiding such constructs as `*, !*.min.js` for every file except minified javascript). If text to find is not entered , and this check box is selected, then IntelliJ IDEA find all files matching the specified mask, regardless of their contents.
Result options	Click the down arrow to reveal the result presentation options. – Skip results tab with one usage - Select this check box to be navigated directly to the found string in the editor, when only one usage is found. The check box is available only in the Find in Path dialog box. – Open in new tab - If selected, sets the search results to be displayed in a separate tab.
Find	– When you click this button in the Find dialog box, IntelliJ IDEA displays the encountered occurrences of the search string in the Find tool window , selects the first occurrence and opens the file with this occurrence in the editor and moves the focus to it.

- When you click this button in the Replace dialog box, IntelliJ IDEA displays the encountered occurrences of the search string in the Find tool window , selects the first occurrence and opens the file with this occurrence in the editor and moves the focus to it.
  At the same time, IntelliJ IDEA opens the Replace Usage dialog box, with the full path to the encountered occurrence in the title bar:



  Do one of the following:
  - To have the selected occurrence replaced, click Replace.
  - To preserve the selected occurrence and move to the next one, click Skip.
  - To have all the occurrences of the search string in the currently active tab replaced, click Replace All in This File.
  - To preserve the occurrences of the search string in the currently active tab (any) and move to the next file, click Skip to Next File.
  - To have all the detected occurrences replaced, click All Files.
  - To switch to the manual mode, click Preview. The Replace Usage dialog box closes and the focus moves to the Find tool window. Do one of the following:
    - Browse through the list of detected occurrences, select the ones you want to replace and then click Replace Selected.
    - To have all the occurrences changed click Replace All.

Preview

Use this tab to check the preview of your search target.

You can press `Up` or `Down` keys to navigate between entries in the preview area without leaving the Text to find field. You also can press `F4` to get to the selected entry in the editor.

Find Usages Dialogs

In this section:

- Find Usages
- Find Usages. Class Options
- Find Usages. Method Options
- Find Usages. Package Options
- Find Usages. Throw Options
- Find Usages. Variable Options

Find Usages

Edit | Find | Find Usages Settings

`Alt+F7`

The dialog also opens when you click in the Show Usages pop-up window which lists all the occurrences of the symbol at caret.

Use this dialog to configure the search procedure and scope when looking for occurrences of the following data:

- Fields, variables and parameters
- Classes, tags, attributes, and references in the HTML, XML, and CSS files
- Symbols at caret.
- Elements of `*.gsp` files in the Grails views.

The search results are displayed in the Find tool window .

Item	Description
Skip results tab with one usage	Select this check box to be navigated directly to the found usage without the Find tool window displayed, when only one usage is found.
Scope	Specify the search scope . Select a pre-defined scope from the drop-down list or click ... to define a custom scope in the Scopes dialog .
Open in new tab	Select this check box to have the results of each search shown on a separate tab of the Find tool window. If the check box is not selected, the search results will be shown on the current tab.

Find Usages. Class Options

Edit | Find | Find Usages

Alt+F7

This section describes the controls for specifying Class Usage Search and Interface Usage Search options in the Find Usages dialog box.

The dialog opens when you click in the Show Usages pop-up window which lists all the occurrences of the symbol at caret.

Item	Description
Find	In this area, specify the objects to search. – Usages - if this check box is selected, the search is performed for all references of the class by its name. – Usages of methods - if this check box is selected, the search is performed for all calls of the selected class methods. – Usages of fields - if this check box is selected, the search is performed for usages of selected class fields. – Derived classes - if this check box is selected, the search is performed for all classes that extend the selected class. Tip The check box is available for class usage search only. – Implementing classes - if this check box is selected, the search is performed for all classes that implement the selected interface. Tip The check box is available for interface usage search only. – Derived interfaces - if this check box is selected, the search is performed for all interfaces that extend the selected interface. Tip The check box is available for interface usage search only.
Options	In this area, configure the search procedure using the following controls: – Search for text occurrences - if this check box is selected, the search is performed in files registered in IntelliJ IDEA. – Skip results tab with one usage - select this check box to be navigated directly to the found usage without the Find tool window displayed, when only one usage is found.
Scope	In this area, specify the scope of search. Select a pre-defined scope from the drop-down list or click the Browse button ... to open the Scopes dialog box, where you can define a custom scope.
Open in new tab	Select this check box to have the results of each search shown in a separate tab of the Find Results window. If the check box is cleared, the search results will overwrite the contents of the current tab.

Find Usages. Method Options

Edit | Find | Find Usages

Alt+F7

This section describes the controls for specifying Method Usage Search options in the Find Usages dialog box.

The dialog opens when you click in the Show Usages pop-up window which lists all the occurrences of the symbol at caret.

Item	Description
Find	In this area, specify the objects to search. – Usages - if this check box is selected, the search is performed for all references of the method by its name. – Overriding methods - if this check box is selected, the search is performed for all methods that override the selected method. – Implementing methods - if this check box is selected, the search is performed for all methods that implement the selected method.
Search for text occurrences	Select this check box to have text contents and comments involved in searching.
Skip results tab with one usage	Select this check box to be navigated directly to the found usage without the Find tool window displayed, when only one usage is found.
Scope	In this area, specify the scope of search. Select a pre-defined scope from the drop-down list or click the Browse button to open the Scopes dialog box, where you can define a custom scope.
Open in new tab	Select this check box have the results of each search shown in a separate tab of the Find Results window. If the check box is cleared, the search results will overwrite the contents of the current tab.

Find Usages. Package Options

Edit | Find | Find Usages

Alt+F7

This section describes the controls for specifying Package Usage Search options in the Find Usages dialog box.

Item	Description
Find	In this area, specify the objects to search. – Usages - if this check box is selected, the search is performed for all references of the package by its name in the source code. – Usages of classes and interfaces - if this check box is selected, the search is performed for all references of classes and interfaces from the selected package by their names in the source code.
Options	In this area, configure the search procedure using the following controls: – Search for text occurrences - if this check box is selected, the search is performed in files registered in IntelliJ IDEA. – Skip results tab with one usage - select this check box to be navigated directly to the found usage without the Find tool window displayed, when only one usage is found.
Scope	In this area, specify the scope of search. Select a pre-defined scope from the drop-down list or click the Browse button ... to open the Scopes dialog box, where you can define a custom scope.
Open in new tab	Select this check box have the results of each search shown in a separate tab of the Find Results window. If the check box is cleared, the search results will overwrite the contents of the current tab.

Find Usages. Throw Options

Edit | Find | Find Usages

Alt+F7

Use this dialog box to find throw usages in the specified scope.

Item	Description
Search for text occurrences	Select this check box to have text contents and comments involved in searching.
Skip results tab with one usage	Select this check box to be navigated directly to the found usage without the Find tool window displayed, when only one usage is found.
Scope	In this area, specify the scope of search. Select a pre-defined scope from the drop-down list or click the Browse button ... to open the Scopes dialog box, where you can define a custom scope.
Open in new tab	Select this check box have the results of each search shown in a separate tab of the Find Results window. If the check box is cleared, the search results will overwrite the contents of the current tab.

Find Usages. Variable Options

Edit | Find | Find Usages

Alt+F7

This section describes Field, Variable, or Parameter usage search options in the Find Usages dialog.

The dialog opens when you click in the Show Usages pop-up window which lists all the occurrences of the symbol at caret.

Item	Description
Skip results tab with one usage	Select this check box to be navigated directly to the found usage without the Find tool window displayed, when only one usage is found.
Scope	In this area, specify the scope of search. Select a pre-defined scope from the drop-down list or click the Browse button to open the Scopes dialog box, where you can define a custom scope.
Open in new tab	Select this check box have the results of each search shown in a separate tab of the Find Results window. If the check box is cleared, the search results will overwrite the contents of the current tab.

Generate Ant Build

Build | Generate Ant Build

Use this dialog box to configure the Ant build generation procedure.

Item	Description
Generate multiple-file ant build (requires ant 1.6 or later to execute)	Click this option to generate separate build files for every module plus one main file. The generated build files for the modules are stored in the directories that correspond to the `.iml` files, and the main file is stored under the project root.
Generate single-file ant build	Click this option to generate one build file for the whole project. This file is stored under the project root.
Backup previously generated files	Click this option to create backup copies of the previously generated Ant build files.
Overwrite previously generated files	Click this option to overwrite the previously generated Ant build files with the new ones.
Output file name	Use this field to enter the name of the output file.
Enable UI forms compilation (requires *javac2* ant task from IntelliJ IDEA distribution)	Select this check box, if you have used IntelliJ IDEA UI forms and need them to be created during the build process.
Use JDK definitions from project files	Select this check box to specify using particular Java SDK's for every module when building a project, as defined in the project settings. Clear the check box to have the Ant script use the Java SDK it runs on.
Inline runtime classpaths	This check box affects Ant build generation for the dependent modules. – If this check box is cleared (by default), then the generated runtime classpath will contain references to the runtime libraries of the dependency module. – If this check box is selected, then the generated runtime classpath will contain copies of references to the libraries of the dependency module.
Use current IntelliJ IDEA instance for `idea.home` property	If this check box is selected, the `idea.home` property, referring to the current IntelliJ IDEA installation, is added to the file `<project>.properties`.

Generate equals() and hashCode() wizard

Code | Generate | context menu | equals() and hashCode()

`Alt+Insert` | context menu | equals() and hashCode()

Use this wizard to generate equals() and hashCode() methods.

Item	Description
Page1	
Template	Use this drop-down list to select a predefined velocity template or click ... to use Templates Dialog .
Accept subclasses as parameter to equals() method	While generally incompliant to Object.equals() specification, accepting subclasses might be necessary for generated method to work correctly with frameworks, which generate Proxy subclasses, like Hibernate.
Use getters during code generation	If this check box is selected, the getters are used in `equals()` instead of direct fields access: `getField()` vs `field` . Click Next to open the next page.
Page 2	
Choose fields to be included in equals()	Select the fields that should be used to determine equality. Each of the selected field's values will be compared, and objects will be considered equal only if all the field values specified here are equivalent. Click Next to open the next page.
Page 3	
Choose fields to be included in hashCode()	Select the fields to generate hash code. Note, that only the fields that were included in the equals() method, can participate in creating hash code. All these fields are selected by default, but you can deselect them, if necessary. Click Next to open the next page.
Page 4	
Select all non-null fields	This page appears if any of the chosen fields are of non-primitive type in order to avoid generation of unnecessary checks. In other words, if a check box for any of these fields is selected, it is presumed that such field never has a null value and such check will not be included into generated methods. Click Finish to complete the wizard and create eqauls() and hashCode() methods.

Templates Dialog

Use this page to select an existing velocity template from the list of available templates or create a custom template.

Item	Description
+	Click this icon to add a new custom template.
−	Click this icon to delete a created custom template.
	Click this icon to copy the existing template.
Equals Template	This area displays a code sample for `equals()` of the selected template.
HashCode Template	This area displays a code sample for `hashCode()` of the selected template.

Generate Groovy Documentation Dialog

Tools | Generate GroovyDoc

Item	Description
Input directory	In this text box, specify the fully qualified path to the directory with the sources to generate documentation for.
Output directory	In this text box, type the fully qualified path to the directory where the generated documentation will be stored. Alternatively, click ... to open the dialog , in which you can select the desired path or create a new directory by clicking .
Private	Select this check box to set the `private` attribute to `true` .
Use	Select this check box to set the `use` attribute to `true` .
Source packages	Use the buttons + and — to create and manage the list of packages containing source files.
Open generated documentation in browser	Select this check box if you want IntelliJ IDEA to automatically open the generated documentation in the browser.

Generate JavaDoc Dialog

Tools | Generate JavaDoc

This dialog invokes JavaDoc utility. The controls of the dialog box correspond to the options and tags of this utility.

Item	Description
Generate JavaDoc scope	Use this area to specify the subset of files, folders, packages etc. for which JavaDoc should be generated. This scope can be the whole project, uncommitted files (when VCS is enabled), current file, custom scope etc.
Include test sources	Select this check box to have the documentation comments for the test files included in the generated JavaDoc.
Include JDK and library sources in `-sourcepath`	If this check box is selected, then paths to the JDK and library sources will be passed to the JavaDoc utility. Refer to documentation for details.
Include link to JDK documentation	If this check box is selected, the references to the classes, packages etc. from JDK will turn into links, which corresponds to using the -link option of the JavaDoc utiity. This check box is only enabled when a link to the online documentation is specified in the Documentation Paths tab of the SDK settings. Refer to JavaDoc documentation for details.
Output directory	In this text box, specify the fully qualified path to the directory where the generated documentation will be stored. Type the path manually or click the Browse button ... to open the Select Path dialog box where you can select the desired location. The specified value is passed to the `-d` parameter of the JavaDoc utility. If the specified directory does not exist in your system, you will be prompted to create it. Note that unless the output directory is specified, the OK button is disabled.
Slider	In this area, specify the visibility level of members to be included in the generated documentation. The available options are: – Private - select this level to have all classes and members included. The level corresponds to the `-private` JavaDoc parameter. – Package - select this level to have all classes and members except private ones included. The level corresponds to the `-package` JavaDoc parameter. – Protected - select this level to have only public and protected classes and members included. The level corresponds to the `-protected` JavaDoc parameter. – Public - select this level to have only public classes and members included. The level corresponds to the `-public` JavaDoc parameter.
Generate hierarchy tree	Select this check box to have the class hierarchy generated. If this check box is cleared, the `-notree` parameter is passed to JavaDoc.
Generate navigator bar	Select this check box to have the navigator bar generated. If this check box is cleared, the `-nonavbar` parameter is passed to JavaDoc.
Generate index	Select this check box to have the documentation index generated. If this check box is cleared, the `-noindex` parameter is passed to JavaDoc.
Separate index per letter	Select this check box to have a separate index file for each letter generated. If this check box is cleared, the `-splitindex` parameter is passed to JavaDoc. The check box is available only if the Generate index check box is selected.
@use	Select this check box to have the use of class and package documented. When selected, the check box corresponds to the `-use` JavaDoc parameter.
@author	Select this check box to have the `@author` paragraphs included. When selected, the check box corresponds to the `-author` JavaDoc parameter.
@version	Select this check box to have the `@version` paragraphs included. When selected, the check box corresponds to the `-version` JavaDoc parameter.
@deprecated	Select this check box to have the `@deprecated` information included. When the check box is cleared, the `-nodeprecated` parameter is passed to JavaDoc.
deprecated list	Select this check box to have the deprecated list generated. When the check box is cleared, the `-nodeprecatedlist` parameter is passed to JavaDoc. The check box is available only if the @deprecated check box is selected.
Locale	In this text box, type the desired locale.
Other command line arguments	In this text box, type additional arguments to be passed to JavaDoc. Use the command line syntax.
Maximum heap size (Mb)	In this text box, type the maximum heap size in Mb to be used by Java VM for running JavaDoc.
Open generated documentation in browser	Select this check box to have the generated JavaDoc automatically opened in the browser.

Generate toString() Dialog

Code | Generate - toString()

Item	Description
Template	Click this drop-down list to choose the template to use.
Settings	Click this button to show Generate ToString() Settings Dialog and set up the behavior and templates of the Generate toString() function.
Toolbar	
	Click this button to represent the fields in ascending or descending alphabetical order.
	Click this button to show classes.
	Click this button to expand all class nodes.
	Click this button to collapse all class nodes.
Class members tree view	Click the members to be included in the generated `toString()` method.
Insert @Override	If this check box is selected, the generated code is prepended with the annotation `@Override`.
OK	Click this button to generate the `toString()` method with the specified settings and close the dialog.
Select None	Click this button to generate the `toString()` method that consists of the method declaration and return statement only.
Cancel	Click this button to discard all changes and close the dialog.

Generate toString() Settings Dialog

Code | Generate - toString() - Settings

On this page:

- Settings tab
- Templates tab
  - Templates toolbar
  - Variables used in Velocity templates

Settings tab

Item	Description
Use fully qualified class name in code generation ($classname)	If this check box is selected, the dumped classnames will include their package names. (The `$classname` variable in the Velocity templates)
Enable getters in code generation ($method)	If this check box is selected, the code generator will have `$methods` variable in the Velocity Macro Language. See example 2 .
Move caret to the generated method	If this check box is selected, the caret scrolls to the generated `toString()` method.
Sort elements	If this check box is selected, the members are sorted in the selected order (ascending or descending).
When method already exists	In this section, choose the default conflict resolution policy: – Ask: Choose this option to ask for confirmation, if a `toString()` method already exists. IntelliJ IDEA shows the dialog box Replace existing toString method. The answer Yes results in generating the new `toString()` method in place of the existing one; the answer No results in creating a duplicate method. – Replace existing: Choose this option to automatically replace the existing `toString()` code. – Generate duplicating method: Choose this option to create a duplicate `toString()` method; so doing, the new method will have the name `toString()`; the existing code will not be erased.
Where to insert?	In this section, choose the place to insert the generated `toString()` method. The possible options are: – At caret. – After equals() and hashCode(): the generated `toString()` method will be inserted after the equals/hashCode, if present in the Java class; otherwise, the new method will be inserted at the current caret position. – At the end of class: the generated `toString()` method will be inserted as the last method.
Exclude	In this section, select the check boxes next to the elements to be excluded from the `toString()` method generation: – Exclude constant fields: If this check box is selected, then any constants won't be a part of the available fields for the code generator. – Exclude static fields: If this check box is selected, then any fields that have static modifiers won't be part of available fields for the code generator. – Exclude transient fields: If this check box is selected, then any fields with transient modifiers won't be be part of the available fields for the code generator. – Exclude enum fields: If this check box is selected, then any fields of the type `enum` (JDK1.5) will not be part of available fields for the code generator. – Exclude logger fields (Log4j, JDK Logging, Jakarta Common Logging): If this check box is selected, then any field that is a either a Log4j Logger, Java JDK Logger or a Jakarta Commons Logger will not be part of the available fields for the code generator. – Exclude fields by name (reg exp): If this check box is selected, then IntelliJ IDEA performs a regular expression matching on the field name. If the result is true, then the field will not be part of the available fields for the code generator. – Exclude fields by type name (reg exp): If this check box is selected, then IntelliJ IDEA performs a regular expression matching on the field type name (fully qualified name). If the result is true, the field will not be part of the available fields for the code generator. – Exclude methods by name (reg exp): If this check box is selected, then IntelliJ IDEA performs a regular expression matching on the method name. If the result is true, the method will not be part of the available methods for the code generator. – Exclude methods by return type name (reg exp): If this check box is selected, then IntelliJ IDEA performs a regular expression matching on the method return type name (fully qualified name). If the result is true, the method will not be part of the available methods for the code generator.

Templates tab

Use this page to view and manage the list of Velocity templates to be used for the `toString()` method generation.

Templates toolbar

Item	Description
+	Click this button to create a new entry to the list of available templates. When the new entry is added, create the corresponding template.
–	Click this button to delete the selected user-created entry from the list of available patterns. Note that pre-defined templates cannot be deleted.
	Click this button to create a copy of an existing template.

Variables used in Velocity templates

Variable	Returns	Description
$classname	String	The name of a class (can be qualified classname, if this is selected in the settings).
$FQClassname	String	@deprecated (use $class.qualifiedName) - The fully qualified name of the class
$fields	java.util.List	List of FieldElement objects
$methods	java.util.List	List of MethodElement objects
$members	java.util.List	List of both FieldElement and MethodElement objects
$member	Element	The Element object
$member.accessor	String	The accessor of a field or method. For a field it is `$field.name` and for a method it is `$method.methodName` .
$member.typeName	String	The classname of the type (Object, String, List etc.)

$member.typeQualifiedName	String	The qualified classname of the type (java.lang.Object, java.lang.String, java.uti.List etc.)
$member.array	boolean	Checks if the type is an array type (either a primitive array or object array).
$member.primitiveArray	boolean	Checks if the type is a primitive array type (int[], short[], float[] etc.)
$member.objectArray	boolean	Checks if the type is an Object array type (Object[], String[] etc.).
$member.stringArray	boolean	Checks if the type is a String array type (String[])
$member.collection	boolean	Checks if the type is assignable from java.util.Collection.
$member.list	boolean	Checks if the type is assignable from java.util.List
$member.map	boolean	Checks if the type is assignable from java.util.Map
$member.set	boolean	Checks if the type is assignable from java.util.Set
$member.primitive	boolean	Checks if the type is a primitive type (int, char, float etc.)
$member.modifierStatic	boolean	Does the type have a static modifier?
$member.modifierPublic	boolean	Does the type have a public modifier?
$member.modifierProtected	boolean	Does the type have a protected modifier?
$member.modifierPackageLocal	boolean	Does the type have a package-local modifier?
$member.modifierPrivate	boolean	Does the type have a private modifier?
$member.modifierFinal	boolean	Does the type have a final modifier?
$member.string	boolean	Is the type assignable from java.lang.String?
$member.numeric	boolean	Is the type assignable from java.lang.Numeric or a primitive type of byte, short, int, long, float, double?
$member.object	boolean	Is the type assignable from java.lang.Object?
$member.date	boolean	Is the type assignable from java.util.Date?
$member.calendar	boolean	Is the type assignable from java.util.Calendar?
$member.boolean	boolean	Is the type assignable from java.lang.Boolean or a primitive boolean?
$field	FieldElement	The FieldElement object
$field.name	String	The name of a field.
$field.modifierTransient	boolean	Does the field have a transient modifier?
$field.modifierVolatile	boolean	Does the field have a volatile modifier?
$field.constant	boolean	Is the field a constant type? (has static modifier and its name is in UPPERCASE only)
$field.matchName(regexp)	boolean	Performs a regular expression matching on a field name.
$field.enum	boolean	Is this field a enum type?
$method	MethodElement	The MethodElement object
$method.name	String	This variable returns one of the following: – The name of the field this getter method covers – The name of the method 'getFoo' when the method does not cover a field
$method.methodName	String	The name of the method (getFoo).
$method.fieldName	String	The name of the field this getter method covers - null if the method is not a getter for a field
$method.modifierAbstract	boolean	Is this method an abstract method?
$method.modifierSynchronized	boolean	Is this method a synchronized method?
$method.returnTypeVoid	boolean	Is this method a void method (does not return anything) ?
$method.getter	boolean	Is this a getter method?
$method.matchName(regexp)	boolean	Performs a regular expression matching on the method name.
$method.deprecated	boolean	Is this method deprecated?
$class	ClassElement	The ClassElement object
$class.name	String	The name of the class
$class.matchName(regexp)	boolean	Performs a regular expression matching on the classname.
$class.qualifiedName	String	The fully qualified name of the class
$class.hasSuper	boolean	Does the class have a superclass? (extends another class - note extending java.lang.Object is not considered having a superclass)
$class.superName	String	The name of the superclass (empty if no superclass)
$class.superQualifiedName	String	The fully qualified name of the superclass (empty if no superclass)
$class.implements(interfaceNames)	boolean	Checks if the class implements the given interface. Checking names of several interfaces can be done by separating the names with commas.
$class.implementNames	String[]	Returns the class names of the interfaces the class implements. An empty array is returned, if the class does not implement any interfaces.
$class.extends(classNames)	boolean	Checks if the class extends any of the given class names. Chcecking several class names can be done by separating the names with commas.
$class.exception	boolean	Is this class an exception class (extends Throwable)?
$class.deprecated	boolean	Is this class deprecated?

$class.enum	boolean	Is this class an enum class?
$class.abstract	boolean	Is this class abstract?
Output variables	The output variables are possible in the Velocity Template (variables are stored in the Velocity Context): Output parameters will be available for the Generate action after the Velocity context has been executed and act upon.	
$autoImportPackages	String	Packagenames that should automatically be imported. Use comma to separate packagenames.

Getter and Setter Templates Dialog

Code | Generate - Getter and Setter

Item	Description
Getter template	Click this drop-down list to choose the template to use. You can click … to open the Template Dialog and add a new getter template or select an existing one.
Setter template	Click this drop-down list to choose the template to use. You can click … to open the Template Dialog and add a new setter template or select an existing one.
	Click this button to represent the fields in ascending or descending alphabetical order.
	Click this button to show classes.
	Click this button to expand all class nodes.
	Click this button to collapse all class nodes.

Template Dialog

Use this page to select an existing velocity template from the list of available templates or create a custom template.

Item	Description
+	Click this icon to add a new custom template.
−	Click this icon to delete a created custom template.
	Click this icon to copy the existing template.
Getter Template	This area displays the selected velocity template and a code sample for `getter`.
Setter Template	This area displays the selected velocity template and a code sample for `setter` of the selected template.

Generate Getter Dialog

Code | Generate - Getter

Item	Description
Template	Click this drop-down list to choose the template to use. You can click … to open the Template Dialog and add a new getter template or select an existing one.
	Click this button to represent the fields in ascending or descending alphabetical order.
	Click this button to show classes.
	Click this button to expand all class nodes.
	Click this button to collapse all class nodes.

Generate Setter Dialog

Code | Generate - Setter

Item	Description
Template	Click this drop-down list to choose the template to use. You can click … to open the Template Dialog and add a new getter template or select an existing one.
	Click this button to represent the fields in ascending or descending alphabetical order.
	Click this button to show classes.
	Click this button to expand all class nodes.
	Click this button to collapse all class nodes.

Gradle Project Data To Import Dialog

Use this dialog to add an external module to a project.

Item	Description
Please select the modules/data to include in the project	Use this area to view the list of available external modules or data for your current project.
	Click this icon to select all the modules from the list.
	Click this icon to clear the list of selected modules.
	Click this icon to see only selected modules.
	If you click this icon, IntelliJ IDEA selects the appropriate modules in case you are unsure about dependencies between the modules.

I18nize Hard-Coded String

`Alt+Enter` - i18nize hard coded string literal

Use this dialog box to extract a hard-coded string literal to the specified `.properties` file. This intention action becomes available, when the Hard-Coded Strings inspection is enabled.

Item	Description
Properties file	In this text box, specify the `.properties` file to store the extracted string literal in. Type the path to the file manually or click the Browse button to open the Choose Properties File dialog, where you can select the desired location using the project tree view or through a search by name. As you type the search string, the suggestion list shrinks to show the matching properties files only.
Update all properties files in resource bundle	Select this check box to have all properties files in the target bundle updated.
Property key	By default, this text box displays the suggested key name, based on the value of the string to be extracted. Accept the default name or type the desired one.
Property value	By default, this field displays the value of the string to be extracted. Accept the default value or type the desired one.
Resource bundle expression	By default, this field displays a resource bundle expression from the resource bundle declaration in the source code. If the resource bundle is not declared in the source code, the field shows an invalid value in red. To improve the situation, define the desired expression. Do one of the following: – Enter an expression of the ResourceBundle type, as described in the section Extracting Hard-Coded String Literals using java.util.ResourceBundle utility class. – Use your own custom utility class. Tip Basic code completion (`Ctrl+Space`) is available in this field.
Edit i18n template	Click this link to open the File and Code Templates dialog box, where you can change the I18nized Expression template to point to the method of a custom utility class that will be used to access a resource bundle. A changed file template is a global setting that affects all projects. If you want to restore defaults, open the File and Code Templates dialog box, find the I18nized Expression template in the Code tab, and click the Reset button .
Preview	This read-only field displays the results of applying the I18nize hard-coded string literal intention action.

Incoming Connection Dialog

The dialog is available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The dialog box opens when you initiate a Zero-Configuration debugging session without previously configuring a Debug Server with path mappings, so the debugger cannot set correspondence between the scripts in the running application and the files in your IntelliJ IDEA project.

The upper psrt of the dialog box is read-only and shows the host and port where the script is executed, the path to the script relative to the server document root, and the absolute path to the script on the server.

In this dialog box, either specify the path mappings manually or import them from the relevant deployment configuration. Learn more at Creating a PHP Debug Server Configuration , Configuring Synchronization with a Web Server , and Zero-Configuration Debugging .

To continue debugging with the imported or manually specified configuration settings, click Accept.

Item	Description
Import mappings from deployment	Choose this option to use the mappings specified in a server access configuration (deployment configuration.). – Project: this read-only field shows the name of the current IntelliJ IDEA project. – Deployment: from this drop-down list, choose the deployment configuration to import the mappings from. If IntelliJ IDEA detects a deployment configuration which seems relevant, the configuration is preselected in the Deployment drop-down list. If IntelliJ IDEA does not detect a relevant configuration: 1. Choose the most suitable configuration from the drop-down list or click … and create a new configuration in the Deployment dialog box that opens, whereupon the new configuration is added to the list. 2. In the Deployment root text box, type the absolute path to the server root folder – Preview: this area shows the absolute path to the project file which corresponds to the currently executed script according to the mappings from the selected configuration.
Manually choose local file or project	When you select this option, IntelliJ IDEA displays the project tree view where you can select a project file and map the currently executed script to it. You can also select and map the entire project.

IntelliJ IDEA License Activation Dialog

To open this dialog: Register on the Welcome screen or Help | Register.

You can evaluate IntelliJ IDEA Ultimate for 30 days. After that period, you need to buy IntelliJ IDEA and activate your license.

The upper part of the dialog reflects your IntelliJ IDEA usage status (e.g. Free evaluation) and, if appropriate, provides related controls (e.g. Buy IntelliJ IDEA).

The license activation options are in the lower part of the dialog under Activate new license via.

- IntelliJ IDEA usage status-related controls
- License activation options

IntelliJ IDEA usage status-related controls

Item	Description
Buy IntelliJ IDEA	Click this button to go to the JetBrains Web site to study the IntelliJ IDEA purchasing options and to buy a license.
Evaluate for free for 30 days	Click this button to start evaluating IntelliJ IDEA.

License activation options

Item	Description
License key	Select this option if you have a license key. Specify your user name (or company name) and the key. (The corresponding information is in your license certificate email.) The recommended way of entering the name and the key is by copying the information from the certificate email and then pasting it into the fields (`Ctrl+C` / `Ctrl+V`). If IntelliJ IDEA rejects the license information, see The key is not accepted for a possible solution.
License server	Select this option if there is the IntelliJ IDEA License Server on your company's intranet. Specify the server URL in the License server address field. For the address to be entered automatically, click the Discover server button. To find out if your company is using the License Server and what its URL should be, contact your system administrator.
JetBrains Account	Select this option if your IntelliJ IDEA license is linked with your JetBrains Account . Specify your JetBrains Account access credentials.

JetBrains Decompiler Dialog

This dialog appears when IntelliJ IDEA tries to decompile the code.

Item	Description
Accept	Click this button if you decide to agree to the specified terms and conditions. In this case, the dialog will not be displayed in the future and the appropriate code will be automatically decompiled.
Decline and restart	Click this button if you do not agree to the specified terms and conditions. In this case, Java Bytecode Decompiler plugin will be disabled and you will need to restart IntelliJ IDEA to implement the change.
Decide later	Click this button if you want to postpone your decision. In this case, the dialog and a window with the decompiled code will close. However, you will be prompted again with the same dialog when IntelliJ IDEA tries to decompile another source code.

Login to IntelliJ Configuration Server Dialog

– Before You Start

– Overview

– Controls

Before You Start

Install and enable the IntelliJ Configuration Server plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Overview

IntelliJ Configuration Server is intended for storing common IDE settings to be used by a team. This ensures that all the team members have unified IDE look and feel.

The dialog box opens:

– When you start up the IntelliJ IDEA for the first time after installing the plugin. In this case, the dialog box contains an additional area On next startup .

– When you click the IntelliJ Configuration Server Status button on the Status bar during a session and then click the Login button in the dialog box that opens.

Use this dialog box to log in to the IntelliJ Configuration Server and specify the Proxy settings to access the Server, if necessary.

Controls

Item	Description
Login	In this text box, type your JetBrains Account login.
Password	In this text box, type your JetBrains account password.
Create or manage your JetBrains account	Click this link to create a JetBrains Account or edit your account if you already have it.
Use HTTP Proxy	Select this check box if you want to access the IntelliJ Configuration Server through a Proxy server.
Host name	In this text box, type the Proxy host name.
Port Number	In this text box, type the Proxy port number.
Proxy Authentication	Select this check box if you want to use a login and password to access the HTTP Proxy server.
Login	In this text box, type your Proxy login.
Password	In this text box, type your Proxy password.
Remember password	If this check box is selected, IntelliJ IDEA remembers your Proxy password and in the future fills in the Password text box automatically when you log in.

On Next Startup Area

The area is displayed in the dialog box only when IntelliJ IDEA starts up for the first time. Use this area to configure how you want to log in to IntelliJ Configuration Server in the future.

The following controls are available:

Item	Description
Show login dialog	Select this option to have the Login to IntelliJ Configuration Server dialog box displayed during the next IntelliJ IDEA startup.
Login silently	Select this option to log in to the Server without asking for login and password.
Do not login	Select this option if you do not plan to log in to the IntelliJ Configuration Server. You can reconfigure this behavior at any time during the session .
Login	Click this button to log in to the IntelliJ Configuration Server.
Skip	Click this button to start an IntelliJ IDEA session without logging to the IntelliJ Configuration Server. You can log in to the Server at any time during the session .

IntelliJ Configuration Server Settings

This dialog box opens, when you click the IntelliJ Configuration Server Status icons / on the Status bar during a session.

Item	Description
Show login dialog	Select this option to have the Login to IntelliJ Configuration Server dialog box displayed during the next IntelliJ IDEA startup.
Login silently	Select this option to log in to the IntelliJ Configuration Server without asking for login and password.
Do not login	Select this option if you do not plan to log in to the IntelliJ Configuration Server. You can reconfigure this behavior at any time during the session .
Login / Logout	Click this button to log in to the IntelliJ Configuration Server.

Manage Project Templates Dialog

Tools | Manage Project Templates

The upper part of the dialog box shows a list of your project templates. When you select a template, its description is shown in the bottom area of the dialog box.

To delete an unnecessary template, select it and click — or press `Alt+Delete`.

Map External Resource Dialog

Specify a URI and select a local XSD or DTD file for the specified URI. (If an XML file references the specified URI, it's validated according to a selected local file.)

Item	Description
URI	Use this field to edit the URI. This may be a URL of an XSD or DTD file (e.g. `http://www.example.org/xsds/example.xsd`) or a namespace URI (e.g. `http://www.example.org`).
Schemas	This tab lists XSD and DTD files available in your project, libraries, SDKs and IntelliJ IDEA installation.
Explorer	This tab implements a file browser .

Non-Project Files Protection Dialog

This dialog appears when you try to edit non-projected files (e.g, library sources, external sources etc.), and protects them from accidental modifications.

Item	Description
These files do not belong to the project	This area displays a non-project file that you are trying to edit.
I want to edit this file anyway	Select this option to disable protection for the listed files.
I want to edit all files in this directory	Select this option to disable protection for the listed files and all files in the same directory.
I want to edit any non-project file in the current session	Select this option to disable protection completely.

All options are effective during current session, once the IDE restarted, protection will be reenabled.

New Action Dialog

Project tool window | Context menu of the target package | New | Action

`Alt+Insert`

Use this dialog to create a new action with the specified identifier, and optionally bind it with a keyboard shortcut that can consist of one or two keystrokes.

Item	Description
ActionId	In this text field, type the identifier of the new action.
Class name	In this text field, specify the name of the class where the new action will be defined. The created stub class contains import statements, and extends `AnAction` .
Name	Specify the name that will identify the new action in the UI.
Add to Group	In this section, select a group where the action will belong (Groups section), and specify its position (Anchor) relative to the other actions in this group (Actions).
Keyboard shortcuts	In the two text fields below, specify the optional primary and secondary keyboard shortcuts, by pressing keyboard keys and their combinations. **Warning!** If you opt to specify the keyboard shortcuts, it is important to use the mouse pointer for clearing the keyboard shortcuts text fields, applying changes and closing the dialog box, because any keystroke will be interpreted as a shortcut.

Open Task Dialog

Tools | Task | Open Task +

Tools | Task | Switch Task - Open Task +

Tasks combo on the main toolbar | Open Task +

`Shift+Alt+T`

Use this dialog box to create a new task or specify the name of an existing task to open. Optionally, specify whether the IntelliJ IDEA's behavior related to opening or creating a task, changes.

On this page:

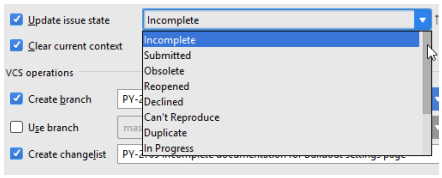
– Suggestion list

– Open Task dialog

Suggestion list

Item	Description
Enter task name	In this text box, type the name of the new task or the name of an existing task to open.
Include closed tasks	Select this check box to include closed tasks into the suggestion list.
	Click this button to open the Servers page of the Settings dialog box.

Open Task dialog

Item	Description
Open task: <task name>	This read-only field shows the name of the task selected from the suggestion list .
Update issue state	Select this check box to choose the task state from the drop-down list:  If this check box is not selected, the drop-down list is disabled.
Clear current context	Select this check box to have the context associated with the current task cleared, when the new/selected task is opened.
VCS operations	

Note This section only appears in the dialog box, when version control is enabled in your project.

Create branch <branch name> from <base branch name>	Select this check box to create a new branch from the existing one.
Use branch	Select this check box to use the existing branch.
Create changelist	Select this check box to have IntelliJ IDEA create a new changelist for the specified task. By default, the text field shows the issue tracker item's description.

Optimize Imports Dialog

Code | Optimize Imports

`Ctrl+Alt+O`

Note that this dialog box appears when you select a Optimize Imports action on the directory.

In this dialog box, specify where you want IntelliJ IDEA to remove unused import statements from, in order to optimize the import procedure.

Item	Description
Only VCS changed files	If this check box is selected, then reformatting will apply only to the files that have been changed locally , but not yet checked in to the repository. This check box is only available for the files under version control.

Override Server Path Mappings Dialog

The dialog box opens when you click the Server paths mappings button in the Debug area of the Run/Debug Configuration: PHP Web Application or Run/Debug Configuration: PHPUnit by HTTP dialog box.

Use this dialog box to map folders on your local machine with folders on the server. These mappings are used during remote debugging and testing as the basis for switching between a test or a stack trace of an exception or assertion and the corresponding source code.

Tip IntelliJ IDEA detects these mappings automatically but still provides you with this possibility to specify them manually.

Item	Description
Local Path on Client	In this text box, specify the absolute path to the desired local folder. Type the path manually or click the Browse button and select the desired folder in the dialog that opens .
Local Path on Server	In this text box, specify the path to the corresponding folder on the server according to the file system used on the server.
Add	Click this button to have a new line added to the list of mappings.
Remove	Click this button to remove the selected mapping from the list.

Play Configuration Dialog

Tools | Play with Playframework

The Tools | Play with Playframework command and the Play Configuration dialog are available only under the following conditions:

- The Playframework Support plugin is enabled. See Enabling and Disabling Plugins .
- A Play application is currently open in IntelliJ IDEA. (In technical terms, the directory `<play_dir>\framework\lib` and the file `<play_dir>\framework\play-<version>.jar` are included in the dependencies of the module that represents your Play application.)

Additionally, the Play Configuration dialog is not shown if the Show on console run option is off in the Play framework settings .

Use this dialog for specifying the Play framework settings such as the Play framework installation directory and the working directory for the `play` command-line utility (the Play console).

Item	Description
Download	Click this link to open the Play framework downloads page which lets you select and download the necessary version of the Play framework. (See which versions are supported by IntelliJ IDEA.)
Home	Specify the Play framework installation directory. Type the path in the field, or click … (`Shift+Enter`) and select the directory in the dialog that opens .
Working directory	Specify the directory from which commands of the `play` command-line utility are to be run. Usually, this is a root directory of your Play application. Type the path in the field, or click … (`Shift+Enter`) and select the directory in the dialog that opens .
Show on console run	Select this check box to be able to see and modify the Play framework settings discussed above each time you access the `play` command-line utility in IntelliJ IDEA.

Print

File | Print

Use this dialog box to specify the text to print and configure the print layout.

In this section:

– Scope
– Settings
– Header and Footer
– Advanced

Scope

Item	Description
File <name>	Select this option to print the file, which is currently selected in the Project view or opened in the editor.
Selected text	Select this option to print the text selected in the editor.
All files in the directory	Select this option to print all files in the current directory.
Include subdirectories	Select this check box to have the files in the subdirectories of the current directory printed as well.

Settings

In this tab, specify the basic print layout settings.

Item	Description
Paper size	From this drop-down list, select the desired paper size.
Font	From the drop-down lists in this area, select the desired font style and size.
Show line numbers	Select this check box to have line numbers printed.
Draw border	Select this check box to have a border printed.
Orientation	In this area, specify the paper orientation. The available options are: – Landscape – Portrait
Style	In this area, specify the style of the printout by selecting the relevant check boxes. The available options are: – Color printing – Syntax printing – Print as graphics

Header and Footer

In this tab, specify the contents and placement of the header and footer.

Item	Description
Text line	In this text box, specify the contents of the header or footer. If necessary, combine plain text with print keywords. By default, IntelliJ IDEA suggests to print the name of a file `$FILE$` in the header and the current page number `$PAGE$` of all pages `$TOTALPAGES$` in the footer. The following print keywords are recognized: – `$FILE$` prints fully qualified file name. – `$PAGE$` – `$DATE$` – `$TIME$` – `$FILENAME$` prints file name without path. – `$TOTALPAGES$`
Placement	Use this drop-down list to specify whether the above line will be printed in the header or in the footer.
Alignment	From this drop-down list, select the desired alignment.
Font	In this area, specify the desired font style and size to print the header and footer text.

Advanced

Item	Description
Wrapping	In this area, configure text wrapping. The available options are: – No wrap – Wrap at word breaks
Margins	Use the text boxes in this area to specify the margins in inches.

Productivity Guide

Help | Productivity Guide

The Productivity Guide shows usage statistics for IntelliJ IDEA features.

The table in the upper part lists the features. Select a feature to see its description in the lower part.

To sort the information, click a cell in the header row.

Item	Description
Feature	The name of the feature.
Group	The group to which the feature belongs.
Used	How many times you used the feature.
Last used	When the feature was last used.

PSI Viewer

Tools | View PSI Structure

Use this viewer to explore internal structure of the various files or fragments of source code, as they are interpreted by IntelliJ IDEA. The dialog box is non-modal and enables you to keep on working with IntelliJ IDEA while being opened.

Item	Description
Show PSI structure for	Use this drop-down list to specify file type, or language constructs to be explored. The set of recognized file types depends on the supported languages and installed plugins.
Show PsiWhiteSpace	If this check box is selected, the generated tree view will contain `PsiWhiteSpace` nodes, corresponding to the spaces in the source code. When you select of clear this check box, the tree view of the PSI structure changes accordingly.
Show Tree Nodes	
Dialect	This drop down list becomes available for the languages that support dialects, for example, SQL, JavaScript etc.
Text	Use this pane to enter source code to be explored. IntelliJ IDEA suggests the following ways to supply code: – Type immediately within the text area. – Paste from clipboard. If you have copied some text from the editor, and then open the PSI viewer, the previous contents of the Text pane is selected, which enables you to overwrite it from the clipboard using `Ctrl+V`, or `Ctrl+Shift+V`. Note that some editing features are also available: removing line at caret `Ctrl+Y`, duplicating text `Ctrl+D`, and adding line with `Shift+Enter`.
PSI Structure	This read-only pane displays the PSI structure tree view, generated on clicking the Build PSI Tree button, according to the file type selected in the Show PSI structure for drop-down list. Navigating though the tree view highlights the corresponding fragments of source code in the Text pane. If currently selected tree node has references, they are also displayed in the References pane.
References	This read-only field shows references to the nodes of the PSI Structure tree view (if any). Unresolved references are shown red; the corresponding fragments of source code are also highlighted with a red frame.
Build PSI Tree	Click this button to generate PSI structure tree view of the code in Text pane, according to the file type selected in the Show PSI structure for drop-down list. If source code in the Text pane has been modified, use this button to refresh the tree view.

Pull Image dialog

For this dialog to be available, the Docker integration plugin must be installed.

Specify the settings for pulling an image from a Docker image repository such as Docker Hub or Quay .

Item	Description
Registry	The URL of the image repository service or a Docker Registry configuration: – If pulling an image doesn't assume logging on to the corresponding server, specify the repository service URL. E.g. registry.hub.docker.com corresponds to public repositories on Docker Hub . – Otherwise, specify the corresponding Docker Registry configuration. Click New to create such a configuration.
New	Click this button to create a Docker Registry configuration. The Docker Registry dialog will open. (A Docker Registry configuration in IntelliJ IDEA represents your Docker image repository user account.)
Repository	The name of the repository (image) to be pulled, e.g. `centos` , `jboss/wildfly` .
Tag	The tag of the image to be pulled.

Push Image dialog

For this dialog to be available, the Docker integration plugin must be installed.

Specify the settings for pushing an image to a Docker image repository such as Docker Hub or Quay .

Item	Description
Registry	Select the Docker Registry configuration to be used. Click New to create such a configuration.
New	Click this button to create a Docker Registry configuration. The Docker Registry dialog will open. (A Docker Registry configuration in IntelliJ IDEA represents your Docker image repository user account.)
Repository	The name of the repository (image) you are pushing.
Tag	The tag for the repository (image) you are pushing.

Recent Changes Dialog

View | Recent Changes

`Shift+Alt+C`

In the Recent Changes popup, use the arrow keys to move in the list, and the `Enter` key to see details and, if necessary, to revert to the previous state.

Refactoring Dialogs

All refactoring dialogs provide similar controls to preview results and perform the refactoring.

Specific options and controls are described in the following topics:

- Change Class Signature Dialog
- Change Signature Dialog for Java
- Change Signature Dialog for ActionScript
- Change Signature Dialog for JavaScript
- Convert Anonymous to Inner Dialog
- Convert to Instance Method Dialog
- Copy Dialog
- Encapsulate Fields Dialog
- Extract Dialogs
- Generify Dialog
- Inline Dialogs
- Inline Method
- Invert Boolean Dialog
- Inline Super Class
- Make Static Dialogs
- Move Dialogs
- Package and Class Migration Dialog
- Pull Members up Dialog
- Push Members down Dialog
- Rename Dialogs
- Replace Constructor with Builder Dialog
- Replace Constructor with Factory Method Dialog
- Replace Inheritance with Delegation Dialog
- Replace Method Code Duplicates Dialog
- Replace Temp with Query Dialog
- Safe Delete Dialog
- Type Migration Preview
- Use Interface Where Possible Dialog
- Wrap Return Value Dialog

Change Class Signature Dialog

Refactor | Change Signature

Ctrl+F6

Use the Change Class Signature dialog to perform the Change Class Signature refactoring in Java.

Manage the formal type parameters using the available controls (see the table that follows). You can add and remove parameters as well as change their order.

Click Refactor to perform the refactoring right away. Click Preview to see the potential changes prior to actually performing the refactoring. (These will be shown in the Find tool window .)

Icon	Shortcut	Description
	Alt+Insert	Use this icon or shortcut to add a parameter. Specify the parameter name and default type in the Name and the Default Value fields respectively.
	Alt+Delete	Use this icon or shortcut to remove the selected parameter.
	Alt+Up	Use this icon or shortcut to move the selected parameter one line up in the list.
	Alt+Down	Use this icon or shortcut to move the selected parameter one line down in the list.

Change Signature Dialog for Java

Refactor | Change Signature

`Ctrl+F6`

Use the Change Signature dialog to perform the Change Method Signature in Java refactoring.

Use the available controls to make changes to the method signature. Specify how the method calls should be handled. Optionally, select the calling methods that the added parameters and exceptions (if any) should be propagated to.

Click Refactor to perform the refactoring right away. Click Preview to see the potential changes prior to actually performing the refactoring. (These will be shown in the Find tool window .)

- Parameters tab
- Exceptions tab

Item	Description
Visibility	Select the method visibility scope (access level modifier) from the list.
Return type	Use this field to modify the method return type. Code completion (`Ctrl+Space`) is available in this field, and also in other fields used for specifying the types.
Name	Use this field to modify the method name.
Parameters	See the description of the Parameters tab .
Exceptions	See the description of the Exceptions tab .
Method calls	Select one of the following options to specify how the method calls should be handled: – Modify. The existing method calls are modified so that the method with the new signature is called. – Delegate via overloading method. The existing method calls don't change. A new overloading method with the old signature is created. This new method calls the method with the new signature.
Signature Preview	In this area, the current method signature is shown. (The information in this area is synchronized with the changes you are making to the method signature.)

Parameters tab

Use the Parameters tab to manage the method parameters.

The available controls let you add new parameters, remove the existing ones, reorder the parameters and also propagate new parameters to the calling methods (see the descriptions that follow).

In addition to that, you can change the type and name for the existing parameters.

To start editing a parameter, just click it. Alternatively, use the `Up` and `Down` arrow keys to move to the parameter of interest and `Enter` to start modifying it.

Item	Tooltip and shortcut	Description
	Add `Alt+Insert`	Use this icon or shortcut to start adding a new parameter. Specify the type, name, and default value in the corresponding fields. (The default parameter value is the value (or the expression) to be passed to the method in the method calls.) If necessary, select the Use Any Var option. As a result, IntelliJ IDEA will search for a variable of the corresponding type near the method call. If such a variable is found, it will be placed in the method call instead of the default value. If more than one variable is found, or the Use Any Var option is not selected, IntelliJ IDEA will use the default value in the call. You can also propagate the parameters you have added to the calling methods.
	Remove `Alt+Delete`	Use this icon or shortcut to delete the selected parameter.
	Up `Alt+Up`	Use this icon or shortcut to move the selected parameter one line up in the list of parameters.
	Down `Alt+Down`	Use this icon or shortcut to move the selected parameter one line down in the list of parameters.
	Propagate Parameters `Alt+G`	Use this icon or shortcut to propagate the added parameters to the calling methods. You can propagate the changes made to the method parameters to any method that directly or indirectly calls the method whose signature you are changing. (There may be the methods that call the current method. These, in their turn, may be called by other methods. You can propagate the changes to any of the methods in such sequences.) In the dialog that opens, select the methods you want the changes to be propagated to. Note that only the selected calling methods and the method calls within them will be affected. That is, the default values will be added into other method calls.

Exceptions tab

Use the Exceptions tab to manage the exceptions thrown by the method.

The available controls let you add new exceptions, remove the existing ones, reorder the exceptions and also propagate new exceptions to the calling methods (see the descriptions that follow).

In addition to that, you can edit the existing exceptions.

To start editing an exception, just click it.

Item	Description
or Alt+Insert	Use this icon or shortcut to add a new exception. Start typing in the field and then select the required exception type from the suggestion list. Note that you can propagate the exceptions you have added to the calling methods.
or Alt+Delete	Use this icon or shortcut to delete the selected exception.
or Alt+Up	Use this icon or shortcut to move the selected exception one line up in the list of exceptions.
or Alt+Down	Use this icon or shortcut to move the selected exception one line down in the list of exceptions.
or Alt+X	Use this icon or shortcut to propagate the added exceptions to the calling methods. You can propagate the exceptions you have added to any method that directly or indirectly calls the method whose signature you are changing. (There may be the methods that call the current method. These methods, in their turn, may be called by other methods. You can propagate the changes to any of the methods in such sequences.) In the dialog that opens, select the methods you want the exceptions to be propagated to.

Change Signature Dialog for ActionScript

Refactor | Change Signature

Ctrl+F6

Use this dialog to change the function signature and to perform other, related tasks.

Item	Description
Return type	Use this field to modify the function return type. Note Code completion is available in this field and also in certain fields of the table that contains the function parameters.
Name	Use this field to modify the function name.
Use the table and the controls to the right of it to manage the function parameters and their properties .	
Type	Use this field to specify the type of a parameter.
Name	Use this field to specify the name of a parameter.
Initializer	A value (or an expression) specified in this field is added to the function definition as the default parameter value.
Default value	A value (or an expression) passed to the function in the function calls.
or Alt+Insert	Use this icon or shortcut to start adding a new parameter. Specify the parameter type , name , initializer , and the default value . Note that you can propagate the parameters you have added to the calling methods.
or Alt+Delete	Use this icon or shortcut to delete the selected parameter.
or Alt+Up	Use this icon or shortcut to move the selected parameter one line up in the list of parameters.
or Alt+Down	Use this icon or shortcut to move the selected parameter one line down in the list of parameters.
or Alt+G	Use this icon or shortcut to propagate the added parameters to the calling methods. You can propagate new function parameters to any function that directly or indirectly calls the function whose signature you are changing. (There may be the functions that call the current function. These functions, in their turn, may be called by other functions. You can propagate new parameters to any of the functions in such sequences.) In the left-hand pane of the Select Methods to Propagate New Parameters dialog, expand the necessary nodes and select the check boxes next to the functions you want the new parameters to be propagated to.
Signature Preview	In this area, the current function signature is shown. (The information in this area is synchronized with the changes you are making to the function signature.)
Refactor	Click this button to perform the refactoring right away.
Preview	Click this button to see the expected changes prior to actually performing the refactoring.

Change Signature Dialog for JavaScript

Refactor | Change Signature

Ctrl+F6

Use this dialog to change the function signature and to perform other, related tasks.

Item	Description
Name	Use this field to modify the function name.
Use the table and the controls to the right of it to manage the function parameters and their properties .	
Name	Use this field to specify the name of a parameter.
Value	A parameter value. If the parameter is a required one, the specified value (or expression) will be passed to the function in the function calls. If the parameter is an optional one, this value will be used in the function body to initialize the parameter.
Optional	Select this check box if the parameter is optional. Your selection will define how the parameter value is used.
or Alt+Insert	Use this icon or shortcut to start adding a new parameter. Specify the parameter name and value . Also, specify whether the parameter is optional . Note that you can propagate the parameters you have added to the calling methods.
or Alt+Delete	Use this icon or shortcut to delete the selected parameter.
or Alt+Up	Use this icon or shortcut to move the selected parameter one line up in the list of parameters.
or Alt+Down	Use this icon or shortcut to move the selected parameter one line down in the list of parameters.
or Alt+G	Use this icon or shortcut to propagate the added parameters to the calling methods. You can propagate new function parameters to any function that directly or indirectly calls the function whose signature you are changing. (There may be the functions that call the current function. These functions, in their turn, may be called by other functions. You can propagate new parameters to any of the functions in such sequences.) In the left-hand pane of the Select Methods to Propagate New Parameters dialog, expand the necessary nodes and select the check boxes next to the functions you want the new parameters to be propagated to.
Signature Preview	In this area, the current function signature is shown. (The information in this area is synchronized with the changes you are making to the function signature.)
Refactor	Click this button to perform the refactoring right away.
Preview	Click this button to see the expected changes prior to actually performing the refactoring.

Convert Anonymous to Inner Dialog

Refactor | Convert Anonymous to Inner

Item	Description
Class name	Specify here the name for the new inner class.
Make class static	Use this option to make the new class static.
Constructor Parameters	In this area select the variables, that will be used as parameters to the inner class constructor.
Move Up/Down	Use these buttons to reorder parameters.

Convert to Instance Method Dialog

Refactor | Convert to Instance Method

Item	Description
Select an instance parameter	Select the class you want the method to belong to after the conversion. All the usages of this class inside the method are replaced with `this`.
Visibility	In this area you can change the visibility scope of the converted method. By default the converted method will have no scope declaration (equivalent to `public`).

Copy Dialog

Refactor | Copy

F5

Use this dialog to specify the settings for the Copy refactoring.

Item	Description
New name	Specify the name of the class, file, package or directory to be created.
To directory	Specify the directory where to create a copy. Select the destination directory from the list, or click ... (Shift+Enter) and select the directory in the Select target directory dialog that opens. (If necessary, you can create a new directory.)
Destination package	When creating a copy of a class: specify in which package the new class should be created. Select the package from the list, or click ... (Shift+Enter) and select the destination package in the Choose Destination Package dialog that opens. (If necessary, you can create a new package.)
Target destination directory	When creating a copy of a class: if there is more than one source root in your module (e.g. one source and one test source root), you have an option of selecting in which of the source roots the new class should be created. Select the destination source root from the list, or click ... (Shift+Enter) and select the source root in the Choose Destination Directory dialog that opens. If the specified destination package doesn't exist in the selected source root, it will be created automatically.
Open copy in editor	Select this check box to automatically open a file, directory or package after it is copied. If you clear this check box, then the file, directory or package that you have copied will not be opened.

Encapsulate Fields Dialog

Refactor | Encapsulate Fields

Item	Description
Fields to encapsulate	In this area select the fields you want to create accessors for. You can accept the default method names or change them at will. If a method with the same signature is already present in the class to be refactored, the Method icon appears to the left of the accessor name, along with the corresponding visibility icon. No new accessor method will be created in this case. The existing method will not be changed and, generally, nothing else will be additionally checked. In this case you have to be attentive whether the existing method actually functions as an accessor. Otherwise, you might need either to choose another name for the accessor or to go back to the editor, change the existing method and only then invoke the Encapsulate Fields refactoring again. If the existing method overrides/implements a parent class' method it will be also marked with the Overrides method or Implements method icon. If the accessor method does not exist in the class to be refactored but, if created it would override/implement a method of the parent class such method will be marked with the Overrides method or Implements method icon only to the left of the accessor name. A new accessor method will be generated. However, this newly created accessor method will actually override or implement the method of the parent class, that might not exactly be what you planned it for. In this case, you might need either to choose another name for the accessor or to go back to the editor, change the existing method and only then invoke the Encapsulate Fields refactoring again.
Get access/Set access	Use this option group to select which accessor methods (Getter, Setter or both) will be created for the selected fields. If one of the check boxes is cleared, the entire corresponding column (Getter or Setter) in the Fields to Encapsulate table is disabled.
Encapsulated Fields' Visibility	Here you can specify the new visibility scope for the selected fields
Options	Select whether you want to use accessors even when field is accessible or not. If the option Use accessors even when the field is accessible is not checked, the references to the desired fields, when the fields are directly accessible, will not change. Otherwise, all references to the desired fields will be replaced with the accessor calls. It also depends on your selection in the options group Encapsulated Fields' Visibility. For example, if you uncheck the option Use accessors even when the field is accessible , and select the private visibility for the fields, the usages of the fields outside the class will change, but within the class they will remain the same.
Accessors' Visibility	In this area select the visibility scope for the created accessor methods.

Extract Dialogs

In this section:

- Extract Constant Dialog
- Extract Class Dialog
- Extract Field Dialog
- Extract Include File Dialog
- Extract Interface Dialog
- Extract Method Dialog
- Extract Method Dialog for Groovy
- Extract Method Object Dialog
- Extract Module Dialog
- Extract Parameter Dialog for ActionScript
- Extract Parameter Dialog for Java
- Extract Parameter Dialog for Groovy
- Extract Parameter Dialog for JavaScript
- Extract Parameter Object Dialog
- Extract Property Dialog
- Extract Superclass Dialog
- Extract Variable Dialog
- Extract Variable Dialog for Sass

Extract Constant Dialog

Refactor | Extract Constant

`Ctrl+Alt+C`

Item	Description
Constant of type	IntelliJ IDEA automatically determines the field type.
Name	Specify here the name for the new constant.
Introduce to class	Select the class in which the constant will be introduced.
Introduce as enum constant	If you've selected an enum class in the Introduce to class field, you can use this option to select, whether you want to introduce this constant as an enum constant, or as a usual field. Otherwise, this option is naturally disabled and does not affect anything.
Visibility	Select the visibility scope for the new field.
Replace all occurrences	Check this option to automatically replace all the occurrences of the selected expression (if the selected expression is found more than once in the class).
Delete variable declaration	Check this option to delete variable declaration.
Annotate field as @NonNls	Check this option to avoid changes during localization. Tip – This field is only available if the project is configured to use annotations, that is, the library `annotation.jar` is added to the project or module. If annotations are not available, this option does not appear in the dialog. – The project language level should be 5.0 to support annotations.

Extract Class Dialog

Refactor | Extract | Delegate

Specify the settings for the extract delegate refactoring .

Item	Description
Name for new class	Specify the name of the class to be created.
Package name	Specify the name of the destination package for the new class.
Members to extract	Select the fields and methods to be extracted to the new class.

Extract Field Dialog

Refactor | Introduce Field

Ctrl+Alt+F

Item	Description
Field of type/ Name	In this text box, specify the name of the new field.
Initialize in	In this area select where the new field will be initialized in.
Visibility	In this area, specify the visibility scope for the new field. The available options are: – Public - if you select this option, the new field will be accessible from anywhere. – Private - if you select this option, the new field will be accessible only from the current class. – Protected - if you select this option, the new field will be accessible from the current class as well as from its inherited and parent classes. – Package local - if you select this option, the new field will be accessible from the classes of the current package only.
Replace all occurrences	Select this check box to have IntelliJ IDEA automatically replace all the occurrences of the selected expression. The check box is enabled only if the selected expression is used more than once in the class. All the found occurrences of the expression are highlighted.
Declare final	Check this option to create a final field.
Delete variable declaration	Check this option to delete variable declaration.

Extract Include File Dialog

Refactor | Extract Include File

Item	Description
Name for extracted include file	In this text box, specify the name of the include file to extract the selected source code to. Tip No extension is allowed.
Extract to directory	In this field, specify the directory to create the include file in. Either accept the predefined directory, or type another one manually, or click the Browse button and choose the desired folder in the dialog that opens .

Extract Interface Dialog

Refactor | Extract Interface

Item	Description
Extract interface from	This read-only field shows the name of the source package that contains the class to extract an interface from.
Extract interface	When this option is selected, IntelliJ IDEA extracts a new interface but does not use it immediately and the source code is not changed.
Interface name	In this text box, type the name for the new interface. The text box is available if the Extract interface option is selected.
Extract interface and use it where possible	Select this option to have an interface extracted and immediately applied to the source code, with the suggested changes displayed in the dedicated tab of the Find tool window.
Rename original class and use interface where possible	Use this option to rename the original class and make it an implementation of the newly created interface.
Rename implementation class to	In this text box, type the new name for the original class. The text box is available if the Rename original class and use interface where possible option is selected.
Package for new interface	In this drop-down list, specify the package for the new interface. If necessary, click the Browse button ... and choose the target package in the dialog that opens .
Members to Form Interface	In this area, specify the methods of the class, as well as final static fields (constants) to be included in the new interface. To have an element included in the interface, select the check box next to it.
JavaDoc/ASDoc	In this area, specify the action to be applied to the inline documentation. The available options are: – As is - select this option to have the inline documentation left where it is. – Copy - select this option to have the inline documentation copied to the extracted interface without removing it from its current location. – Move - select this option to have the inline documentation moved to the extracted interface and delete it from its current location.

Extract Method Dialog

Refactor | Extract Method

Ctrl+Alt+M

Item	Description
Name	In this text box, specify the name of the function or method to be generated on the basis of the selected source code.
Visibility	In this area, specify the visibility scope of the method to be generated.
Declare static	Select this check box to have a static method created. Tip If the new method cannot be declared as static, or, vice-versa, can be created only as a static method, the Declare Static check box is disabled.
Declare varargs	Select this option if you want to declare varargs instead of the array.
Fold parameters	Select this option to fold the parameters, for example, if you have an array, like `int[] a = new int[i]`, and you want `a[i]` to be passed as a whole to the newly created method.
Extract chained constructor	Use this option to extract chained constructor from the constructor body, replacing the original code with `this`.
Output variable(s)	This read-only text box displays the name of the variable through which the output of the new method/function will be passed to the calling method/function. Depending on your choice in the Return output variable(s) through area, this variable either will be used in a return statement or will be declared as the passed by reference parameter of the new method/function.
Return output variable(s) through	In this area, specify the way in which the new method or function will return the output variables to the callee.. – Return statement - select this option to have the output variables returned by value. If the Output variable(s) read-only field shows exactly one output variable, it will be used as the return value. If the selection outputs several variables, these variables will be returned as an array. – Parameter(s) passed by reference - select this option to have the output variables returned by reference. IntelliJ IDEA will generate a method/function without a return statement. Instead, the output variables will be added to the set of input parameters in the method/function declaration. The names of these variables will be prepended with an ampersand `&`. The area is available only when refactoring is invoked in the PHP context.
Parameters	In this area, select parameters to be passed to the new method/function. Tip If any parameter that is critical for the functionality of the new method is not selected, IntelliJ IDEA will be unable to proceed with the refactoring.
Move Up/Down	Use these buttons to change the order of the parameters.
Signature preview	In this read-only field, view the declaration of the new method.

Extract Method Dialog for Groovy

Refactor | Extract Method

Ctrl+Alt+M

Item	Description
Visibility	In this area, specify the visibility scope of the method to be generated.
Name	In this text box, specify the name of the function or method to be generated on the basis of the selected source code.
Specify return type explicitly	This check box is available if you invoke refactoring from the method of a Groovy class. Select this check box to return a data type of the value explicitly.
Use explicit return statement	This check box is active if the method returns a value. You can omit `return` keyword if it is the last return statement in the method. If you select this check box the keyword is returned.
Parameters	In this area, select parameters to be passed to the new method/function. Tip If any parameter that is critical for the functionality of the new method is not selected, IntelliJ IDEA will be unable to proceed with the refactoring.
Move Up/Down	Use these buttons to change the order of the parameters.
Signature preview	In this read-only field, view the declaration of the new method/function.

Extract Method Object Dialog

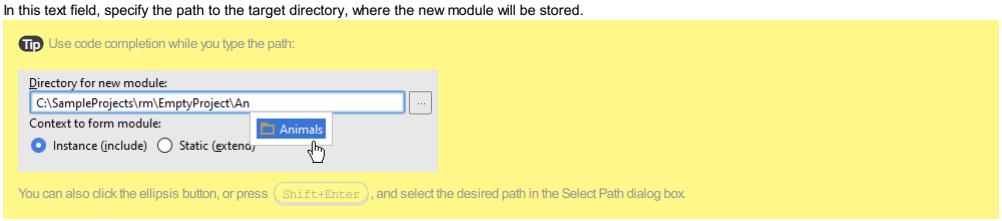
Refactor | Extract Method Object

Use this refactoring to extract method object, when Extract Method refactoring is not applicable, because of multiple return values.

Item	Description
Create inner class	Select this option, if you want to extract method object to an inner class. All the local variables will become fields on that class. Specify also name for the class and visibility scope. You can make the class static, if needed.
Create anonymous class	Select this option to an object with the corresponding method.
Parameters	In this area select the variables that will be used as a parameters. Use Move Up/ Move Down buttons to reorder parameters/

Extract Module Dialog

Refactor | Extract Module

Item	Description
Extract module from	This read-only field displays the name of Ruby class, from which a module should be extracted.
Module name	In this text field, type the name of the target module. The module name should a proper Ruby constant.
Directory for new module	In this text field, specify the path to the target directory, where the new module will be stored. 
Context to form module	Click one of the radio buttons (Instance or Static) to define the way the new module will be used in a Ruby class.
Members to form module	This area displays the list of members detected in the original class. Select the check boxes next to the members to be included in the new module. Note that static methods are disabled when Instance context is selected, and vice versa.

Extract Parameter Dialog for ActionScript

Refactor | Extract | Parameter

Ctrl+Alt+P

Use this dialog to specify the options and settings related to the Extract Parameter refactoring in ActionScript .

Item	Description
Type	Specify the type of the new parameter. Usually, you don't need to change the type suggested by IntelliJ IDEA.
Name	Specify the name for the new parameter.
Value	The expression to be replaced with the new parameter. Initially, this field contains the expression that you have selected. Normally, this initial value does not need to be changed.
Optional parameter	If you want the new parameter to be an optional parameter, select this check box. For information about required and optional parameters, see the discussion of function parameters in Flex/ActionScript documentation .
Replace all occurrences	Select this option to replace all the occurrences of the selected expression within the function.

Extract Parameter Dialog for Java

Refactor | Extract | Parameter

`Ctrl+Alt+P`

Refactor | Extract | Functional Parameter

`Ctrl+Shift+Alt+P`

Specify the settings for extracting a parameter. See Extract Parameter in Java and Extract Functional Parameter .

Item	Description
Parameter of type	Specify the type of the new parameter. Usually, you don't need to change the type suggested by IntelliJ IDEA.
Name	Specify the name for the new parameter.
Replace all occurrences	Select this option to replace all the occurrences of the selected expression within the method.
Declare final	Select this option to declare the parameter `final` .
Delegate via overloading method	Select this option to keep the existing method calls unchanged. As a result, a new overloading method will be created and the new parameter will be added to the definition of this method. The method with the old signature will be kept. The call to the new method will be added to the old method. The necessary value (or expression) will be passed to the new method in this call.
The following options are available if the expression contains a direct call to a class field that has a getter.	
Do not replace	Select this option to use a direct call to the field regardless of the scope context.
Replace fields inaccessible in usage context	Select this option to use a call to the getter only where a field is directly inaccessible.
Replace all fields	Select this option to use a call to the getter.
The following options are available when the refactored expression contains local variables.	
Delete variable definition	Select this option to remove the definition of a local variable.
Use variable initializer to initialize parameter	Select this option to use the default value of the selected variable in the corrected method calls. **Warning!** It is strongly recommended that you select both check boxes. Otherwise, the code may become uncompilable.

Extract Parameter Dialog for Groovy

Refactor | Extract | Parameter

Ctrl+Alt+P

Use this dialog to specify the options and settings related to the Extract Parameter refactoring in Groovy .

Item	Description
Type	Specify the type of the new parameter. Usually, you don't need to change the type suggested by IntelliJ IDEA.
Name	Specify the name for the new parameter.
Declare final	Select this option to declare the parameter `final`.
Delegate via overloading method	Select this option to keep the existing method calls unchanged. As a result, a new overloading method will be created and the new parameter will be added to the definition of this method. The method with the old signature will be kept. The call to the new method will be added to the old method. The necessary value (or expression) will be passed to the new method in this call.
Remove parameter <name> no longer used	Select this check box to remove a parameter.
Use explicit return statement	This check box is active if the method returns a value. You can omit `return` keyword if it is the last return statement in the method. If you select this check box the keyword is returned.
Parameters	In this area, select parameters to be passed to the new method/function. Tip If any parameter that is critical for the functionality of the new method is not selected, IntelliJ IDEA will be unable to proceed with the refactoring.
Move Up/Down	Use these buttons to change the order of the parameters.
Signature preview	In this read-only field, view the declaration of the new method/function.

Extract Parameter Dialog for JavaScript

Refactor | Extract | Parameter

Ctrl+Alt+P

Use this dialog to specify the options and settings related to the Extract Parameter refactoring in JavaScript .

Item	Description
Type	Specify the type of the new parameter. Usually, you don't need to change the type suggested by IntelliJ IDEA.
Name	Specify the name for the new parameter.
Value	The expression to be replaced with the new parameter. Initially, this field contains the expression that you have selected. Normally, this initial value does not need to be changed.
Optional parameter	If this option is selected, the new parameter is assigned a value in the function body. The value corresponds to that currently set in the Value field. The calls to the function do not change. If this option is not selected, all the function calls change according to the new function signature. The value specified in the Value field is added to the function calls and thus is passed to the function. No explicit value assignment for the new parameter is added to the function body.
Replace all occurrences	Select this option to replace all the occurrences of the selected expression within the function.

Extract Parameter Object Dialog

Refactor | Extract | Parameter Object

Use this refactoring to create a wrapper class around some selected parameters of a method, or use a compatible existing class as a wrapper.

Item	Description
Method to extract parameters from	This read-only field shows the name of the selected method.
Parameter Class	Use this section to specify whether you want to create a new wrapper class or use an existing one.
Create new class	Click this radio-button to move parameters of a method to a new class. If this option is selected, specify the class and destination package name in the fields below. By default, current package name is displayed. You can type a different package name in the text field, or click the ellipsis button and select the destination package from the tree view. If the desired package doesn't exist, click to create a new one.
Create inner class	Click this radio-button to move parameters of a method to an inner class. If this option is selected, specify the class name in the field below.
Use existing class	Click this radio-button to move parameters of a method to an existing class of your choice. You can type the fully-qualified class name in the text field, or click the ellipsis button and choose the desired class in the Select parameter class dialog box. Note that you can select the desired wrapper class both from the project and non-project classes.
Parameters to extract	Use check boxes in this section to select which parameters of a method should be extracted into separate object.
Move Up / Move Down	Use these buttons to move the selected reorder parameters in the list.
Keep method as delegate	If this check box is selected, the original method will be preserved as a delegate to the newly created method.

Extract Property Dialog

Refactor | Introduce Property

`Ctrl+Alt+V`

This refactoring is only available for `pom.xml` files.

Item	Description
Name	In this field, specify the name you want to assign to a new property. You can either type this name manually, or select one of the values suggested by IntelliJ IDEA.
Project	Select the `pom.xml` where the new property will be declared.

Extract Superclass Dialog

Refactor | Extract Superclass

Item	Description
Extract superclass from	This read-only field shows the name of the source package that contains the class to extract a superclass from.
Extract superclass	When this option is selected, IntelliJ IDEA extracts a new superclass but does not use it immediately and the source code is not changed.
Superclass name	In this text box, type the name for the new superclass. The text box is available if the Extract superclass option is selected.
Extract superclass and use it where possible	Select this option to have a superclass extracted and immediately applied to the source code, with the suggested changes displayed in the dedicated tab of the Find tool window.
Rename original class and use superclass where possible	Use this option to rename the original class and make it an implementation of the newly created superclass.
Rename original class to	In this text box, type the new name for the original class. The text box is available if the Rename original class and use superclass where possible option is selected.
Package for new superclass	In this drop-down list, specify the package for the new superclass. If necessary, click the Browse button and choose the target package in the dialog that opens .
Members to Form Superclass	In this area, specify the members of the class to be moved or delegated to the new superclass. To have an element included in the interface, select the check box next to it.
Make Abstract	Select this option to leave the method implementation within the current class, and declare it abstract in the extracted superclass.
JavaDoc/ASDoc for abstracts	In the JavaDoc/ASDoc area, specify the action to be applied to the inline documentation. The available options are: – As is - select this option to have the inline documentation left where it is. – Copy - select this option to have the inline documentation copied to the extracted superclass without removing it from its current location. – Move - select this option to have the inline documentation moved to the extracted superclass and delete it from its current location.

Extract Variable Dialog

Refactor | Extract Variable

`Ctrl+Alt+V`

Generally, this dialog is used to extract new variables. Depending on the language, there may be additional options, for example:

- In Java, you can declare the new variable `final`.
  To make this dialog accessible for Java, you have to disable in-place refactoring in the editor settings.
- In JavaScript, you can select to extract a (global) variable or a constant. For JavaScript 1.7 or a later version, there is also an option of extracting a local variable.
- In ActionScript, you can choose to extract a constant rather than a variable, and specify the variable type.

Item	Description	Language
Type	Select the type of the new variable.	ActionScript
Name	Specify the name for the new variable.	All
Replace all occurrences	Select this option to replace all the occurrences of the selected expression (if more than one occurrence of the selected expression is found).	All
Declare final	Select this option to declare the new variable `final`.	Java
Var kind	Choose to extract a (global) variable, a constant, or a local variable: – extract var - select this option to extract a global variable. – extract constant - select this option to extract a constant. – extract local variable - select this option to extract a local variable. The extract local variable option is available only for JavaScript 1.7 or a later version.	JavaScript
Make constant	Select this option to extract a constant rather than a variable.	ActionScript

Extract Variable Dialog for Sass

Refactor | Extract Variable

Ctrl+Alt+V

Use this dialog box to replace a Sass expression with a variable.

Item	Description
Name	Specify the name for the new variable.
Place for declaration	Select the place in the source code, where the new variable will be declared. The declaration can be global (a variable is declared in the beginning of a Sass file and is available throughout the whole file), or local (a variable is declared immediately before use, and is available in the current block only).
Replace all occurrences	Select this option to replace all the occurrences of the selected expression (if more than one occurrence of the selected expression is found).

Generify Dialog

Refactor | Generify

Item	Description
Drop obsolete casts	If this option is checked, IntelliJ IDEA analyzes whether the parameter cast cases are changed by refactoring. If the resulting parameter type is similar to the obsolete one, the cast statement is removed.
Leave Object-parameterized types raw	Check this option to make objects, that have `java.lang.Object` as a parameter, raw.
Perform exhaustive search	Check this option to perform search in all nodes.
Generify Objects	Check this option to transform the `java.lang.Object` objects into the type, they are actually used for.
Produce wildcard types	Check this option to produce wildcard types where possible (expressions like `List<? extends String>`).
Preserve raw arrays	If this check box is selected, the arrays are not changed to the arrays with parameterized types. Otherwise, the arrays will be transformed to parameterized type. Clearing this check box can be risky and result in uncompilable code.

Inline Dialogs

Refactor | Inline

<context menu of a selection> | Refactor | Inline

Ctrl+Alt+N

– Inline Variable dialog
– Inline to Anonymous Class dialog

Inline Variable dialog

The Inline Variable refactoring allows you to replace a redundant variable with its value. See examples .

To access the Inline Variable dialog box through a menu item or the keyboard shortcut, position the cursor at the variable to be inlined. The dialog box does not contain any controls but just displays the following message:

```
Inline variable <variable name>? (<the number of variable occurrences>)
```

Inline to Anonymous Class dialog

Item	Description
All references and remove the class	Select this radio button to replace all the class references with its code and remove the class.
This reference only and keep the class	Use this option to replace only the current class reference.
Search in comments and strings	Select this check box to display the usages of methods in comments and strings in the Refactoring Preview tool window. Note This option will force the Refactoring Preview tool window to open even if you click the Refactor button instead of the Preview button.
Search for text occurrences	Check this option to apply the changes to non-java files (such as documentation, HTML, JSP and other files included in the project).

Inline Method

Refactor | Inline

Inline Method refactoring results in placing method's body into the body of its caller(s). You can opt to:

- inline all occurrences of the method, and delete the method
- inline only a single occurrence, and retain the method

Item	Description
Inline all invocations and remove the method	Select this radio button to replace all the method calls with its code and remove the method.
Inline this invocation only and keep the method	Use this option to replace only the current method call.
Search in comments and strings	Select this check box to display the usages of methods in comments and strings in the Refactoring Preview tool window. Note This option will force the Refactoring Preview tool window to open even if you click the Refactor button instead of the Preview button.
Search for text occurrences	Check this option to apply the changes to non-java files (such as documentation, HTML, JSP and other files included in the project).

Invert Boolean Dialog

Refactor | Invert Boolean

Item	Description
Name of inverted method/field	Specify the name for the inverted method or field.

Inline Super Class

Refactor | Inline

Inline Super Class refactoring results in pushing superclass' methods into the class where they are used, and removing the superclass.

Item	Description
JavaDoc for inlined members	In this area you can select an action that can be applied to JavaDoc. You can choose from the following options: – As is - lets you leave the inline documentation where it is. – Copy - lets you copy the inline documentation to the destination superclass without removing it from its current location. – Move - lets you move the inline documentation to the destination superclass and delete it from its current location.
Inline all references and remove the class	Select this radio button to replace all the super class references with its code and remove the super class.
Inline this reference only and keep the super class	Select this radio button to replace only the current super class reference.

Make Static Dialogs

Refactor | Make Static

Use this refactoring to convert an inner class or an instance method into a static one.

– Make Class Static Dialog

– Make Method Static Dialog

Make Class Static Dialog

Item	Description
Replace instance qualifiers with class references	Specify whether you want to replace instance qualifiers with class references or not.

Make Method Static Dialog

Item	Description
Add object as a parameter with name	Select this check box, if you want to pass the whole referenced object as a parameter to the method, then specify the name for the parameter in the field below.
Add parameters for fields	Select this check box to pass the referenced fields/variables as parameters to the method, then select the appropriate fields in the list.
Move Up/Move Down	Use this buttons to reorder parameters in the list.
Replace instance qualifiers with class references	Specify whether you want to replace instance qualifiers with class references or not. This check box is available, if the method does not contain any references to fields or instance variables.

Move Dialogs

Refactor | Move

F6

In this section:

- Move Class Dialog
- Move Directory Dialog
- Move Package Dialog
- Move Instance Method Dialog
- Move Inner to Upper Level Dialog for Java
- Move Inner to Upper Level Dialog for ActionScript
- Move Members Dialog
- Move File Dialog
- Move Namespace Dialog

Move Class Dialog

Refactor | Move

F6

The Move Class refactoring dialog box is invoked for the classes selected in the Project view, or opened in the editor.

Item	Description
To package	Specify the destination package. Click the ellipsis button, and select the target package in the Choose Destination Package dialog that shows a tree view of all packages within the project.
Make inner class of	If you want to make your class inner, specify the target class. Click the ellipsis button and select the target class in the Choose Class dialog. This option is not available for ActionScript.
Search in comments and strings	Select this option to apply the changes to comments and strings.
Search for text occurrences	Select this option to apply the changes to documentation, HTML, JSP and other files included in your project.
Search for references	Select this check box to have the changes applied to the references to the file in question. This option is available only for Move File or Move Package refactorings.
Move to another source folder	If this option is selected, you can select the target root, where the destination package will be located. If the option is not selected, only the current root is used. This option is disabled for the modules that contain a single source root.
Target destination directory	When the dialog box opens, the field shows the path to the folder where the file that implements the class to move is currently stored. The path is displayed in the format `...\<project root folder>\<path to the current namespace folder relative to the project root>`. The path is updated automatically as you specify the namespace to move the class to. However, if you are going to move a class to a non-existing namespace under another parent namespace, IntelliJ IDEA will not suggest the proper folder unless you appoint a root folder for your namespace structure by marking the relevant folder as Sources on the Directories page of the Settings dialog box, see Directories for details. Do one of the following: – Accept the preselected path displayed in the field. – Choose another path from the list. All of them are evaluated from the namespace root or from the current directory, so it is safe to choose any of them. – Click ... and select a folder in the dialog box that opens. – Press F2 and edit the preselected path. Keep in mind that this may cause problems with automatic loading in the future.
Open moved in editor	Select this check box to open the moved class in the editor.

Move Directory Dialog

The Refactor | Move menu command invokes the Move Directory dialog box when a directory is selected in the Project tool window.

Item	Description
To directory	Specify the destination directory. Type the path manually or click ... and select the target directory in the dialog that opens .
Search for references	Select the check box to find and update the references to the directory being moved.
Open moved files in editor	Select this check box to see the files of the moved directory in the editor.

Move Package Dialog

Refactor | Move

F6

Item	Description
Select refactoring	Specify which action you want to perform: move package to another package , move directory to another source root , or to another directory .
This dialog box opens when Move package to another package option has been selected.	
To package	Specify the destination package. Click the ellipsis button, and select the target package in the Choose Destination Package dialog that shows a tree view of all packages within the project.
Search in comments and strings	Select this option to apply the changes to comments and strings.
Search for text occurrences	Select this option to apply the changes to documentation, HTML, JSP and other files included in your project.
Search for references	Select this check box to have the changes applied to the references to the file in question. This option is available only for Move File or Move Package refactorings.
Move to another source folder	If this option is selected, you can select the target root, where the destination package will be located. If the option is not selected, only the current root is used. This option is disabled for the modules that contain a single source root.
This dialog box opens when Move directory to another source root option has been selected.	
Select source root	Choose the target source root from a tree view of source roots configured in your project.
This dialog box opens when Move directory to another directory option has been selected.	
To directory	Use this field to specify the destination directory. Click the ellipsis button, and select the target directory in the Select Path dialog box.

Move Instance Method Dialog

Refactor | Move

F6

Item	Description
Select an instance parameter	Select the target class.
Visibility	Select visibility scope for the method to be moved.

Move Inner to Upper Level Dialog for Java

Refactor | Move

F6

Move Inner Class refactoring dialog box is invoked for inner classes selected in the Structure view or opened in the editor.

Item	Description
Class name	Use this field to rename the moved inner class, if needed.
Package name	Specify the name of the package for the class to be moved to.
Pass outer class' instance as a parameter	Select this check box, if you want the moved class to preserve access rights to its former outer class.
Parameter name	Specify here the name for the outer class' instance passed as a parameter.
Search in comments and strings	Select this option to apply the changes to comments and strings.
Search for text occurrences	Select this option to apply the changes to documentation, HTML, JSP and other files included in your project.
Search for references	Select this check box to have the changes applied to the references to the file in question. This option is available only for Move File or Move Package refactorings.

Move Inner to Upper Level Dialog for ActionScript

Refactor | Move

F6

This dialog opens when moving an out-of-package class, function, variable, constant or namespace into a package. You can specify the destination package and other, associated settings.

The out-of-package entities being moved are referred to in the dialog as inner entities, for example, an inner class, an inner function, etc.

This dialog can be accessed from:

- The editor when the cursor is within the item (class, function, etc.) that you are going to move.
- The Structure tool window when the corresponding item is selected there.

Item	Description
<Entity> name	If necessary, change the name of the entity (class, function, etc.) that you are moving.
Package name	Specify the destination package. Select the package from the list, or click ... and select the package in the Select Destination Package dialog.
Search in comments and strings	Select this option to apply the changes to comments and strings.
Search for text occurrences	Select this option to apply the changes to documentation, HTML, JSP and other files included in your project.
Move to another source folder	If this option is selected, you can select the target root, where the destination package will be located. If the option is not selected, only the current root is used. This option is disabled for the modules that contain a single source root.

Move Members Dialog

Refactor | Move

F6

Move Member refactoring dialog box is invoked for static members selected in the Structure view, or in the editor.

Item	Description
Move members from	This read-only field displays the fully qualified name of the source class containing members to be moved.
To (fully qualified name)	Specify the fully qualified name of the target class.
Move as enum constant if possible	Select this option to move member as enum constant, if it is possible.
Members to be moved (static only)	This table shows all static members detected in the specified class. Select check boxes next to the members you want to move.
Visibility	Specify visibility level. You can either specify it explicitly, or select Escalate to automatically raise it to a necessary level.

Move File Dialog

Refactor | Move

F6

The Refactor | Move menu command invokes one of the following dialog boxes depending on the selected item to be moved:

- The Move File dialog box is invoked when a file is selected in the Project tool window or opened in the editor.
- The Move File dialog box is invoked when a file opened in the editor.

Item	Description
To package	Specify the destination package. Type the path manually or click ... and select the target package in the dialog that opens .
Search in comments and strings	Select the check box to find and update the comments and strings to the file being moved.
Open moved in editor	Select this check box to see the moved file in the editor.

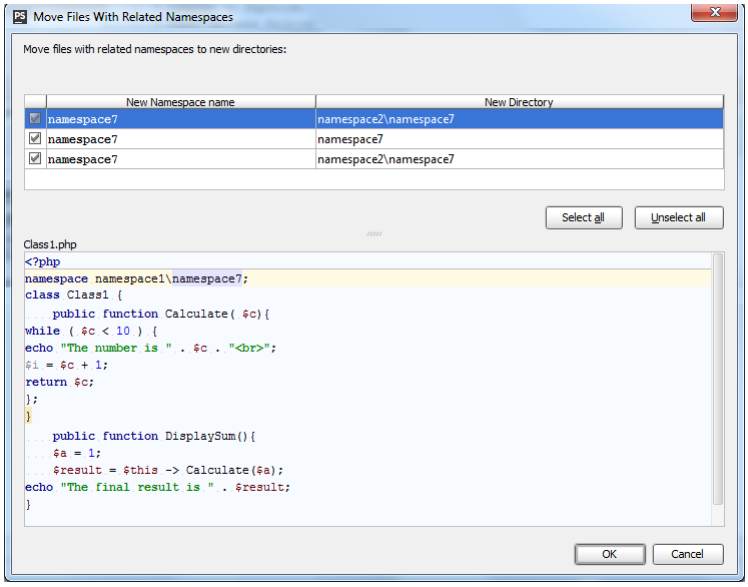
Move Namespace Dialog

The dialog box is available only if the PHP plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The dialog box opens when you select a PHP namespace to be moved and choose Refactor | Move on the main menu or on the context menu of the selection.

IntelliJ IDEA supposes that the namespaces in your project are arranged in compliance with the PSR0 standard and enforces you to retain the folder structure and the namespace hierarchy in accordance with this standard when moving namespaces.

When you specify a namespace to move a namespace to, IntelliJ IDEA automatically updates the Target Destination Directory field, which shows the path to the folder that corresponds to the namespace in question.

Item	Description
New Namespace Name	When the dialog box opens, the field shows the fully qualified name of the selected namespace. Specify the new namespace name. Use only backslashes (`\`) as namespace separators.
Target Destination Directory	When the dialog box opens, the field shows the path to the folder which corresponds to the current namespace. The path is displayed in the format `...\<project root folder>\<path to the current namespace folder relative to the project root>` . The path is updated automatically as you specify the new namespace name. However, if you are going to move a namespace to another parent namespace, IntelliJ IDEA will not suggest the proper folder unless you appoint a root folder for your namespace structure by marking the relevant folder as Sources on the Directories page of the Settings dialog box, see Directories for details. Do one of the following: – Accept the preselected path displayed in the field. – Choose another path from the list. All of them are evaluated from the namespace root or from the current directory, so it is safe to choose any of them. – Click ... and select a folder in the dialog box that opens. – Press F2 and edit the preselected path. Keep in mind that this may cause problems with automatic loading in the future.
Refactor	Click this button to open the Move Files with Related Namespaces dialog box and specify the classes and files to be moved to the new namespace and the new folder. The upper pane of the dialog box lists the destination namespaces and folders for classes and files related to the namespace in question. Each item in the list corresponds to a class/file. When you move the cursor to an item, the bottom pane shows the contents of the file related with it.  – To have a class and the corresponding file moved to the destination namespace and the destination folder, select the check box next to the namespace/folder. – To add all the items to the list or remove all of them from the list, click Select All or Unselect All respectively.

Package and Class Migration Dialog

Refactor | Migrate

Item	Description
Select migration map	Select the desired migration map from the drop-down list. By default, only Swing 1.0.3.to 1.1 migration map is present.
Edit	Click this button to change the selected migration map, using the Edit Migration Map dialog.
New	Click this button to create your own migration map, using the Edit Migration Map dialog.
Remove	Click this button to delete the selected migration map from the list.

Pull Members Up Dialog

Refactor | Pull Members Up

Use this dialog to pull selected members up to the selected superclass or interface.

Item	Description
Pull Up Members of <class_name> to	Select the destination object (interface or superclass) .
Members to be pulled up	In this area, select the members you want to move.
Make abstract	Select this check box to move the selected method as abstract.
JavaDoc for abstracts	In this area, select the action to be applied on the JavaDoc. − To leave it where it is, select the As is option. − To copy it to the destination superclass (subclass) or interface (subinterface), select the Copy option. − To move it to the destination subclass or subinterface, select the Move option.

Push Members Down Dialog

Refactor | Push Members Down

Use this dialog to push selected members down to immediate subclass.

Item	Description
Members to be pushed down	In this area, select the members you want to move.
Keep abstract	Select this check box to preserve moved method as abstract.
JavaDoc for abstracts	In this area, select the action to be applied on the JavaDoc. − To leave it where it is, select the As is option. − To copy it to the destination superclass (subclass) or interface (subinterface), select the Copy option. − To move it to the destination subclass or subinterface, select the Move option.

Rename Dialogs

Refactor | Rename

Shift+F6

The Rename dialogs let you perform the Rename refactoring for various entities such as classes, fields, methods, packages, etc. You can select where the changes to occurrences of the entity name are required and where not.

- Rename Dialog for a Class or an Interface
- Rename Dialog for a Directory
- Rename Dialog for a Field
- Rename Dialog for a File
- Rename Dialog for a Method
- Rename Dialog for a Package
- Rename Dialog for a Parameter
- Rename Dialog for a Table or Column
- Rename Dialog for a Variable

Rename Dialog for a Class or an Interface

Refactor | Rename

`Shift+F6`

Use this dialog to rename a class or an interface.

In addition to renaming the class or the interface itself, IntelliJ IDEA can also look for the usages of the class or the interface name. If found, the changes you are making to the class or the interface name can also be applied to these usages.

The usages are assigned to different categories which correspond to the options which you can turn on and off.

Note that regardless of the options selected, the search scope (that is, the places where the name occurrences are looked for) is always limited to the current entity (file, class, etc) and the entities that the current one depends on.

Item	Description
Rename <class or interface> and its usages to	Specify a new name for the class or the interface.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the class or the interface name within comments and string literals in your source code files.
Search for text occurrences	If this option is on, IntelliJ IDEA will look for occurrences of the class or the interface name in the files that don't contain source code. These may be the text files, properties files, HTML files, documentation files, etc.
Rename variables	If this option is on, IntelliJ IDEA will look for occurrences of the class or the interface name in the names of variables. Note that only the variables that have the type of the class or the interface you are renaming will be taken into account. To illustrate, let's assume that you are renaming the class `MyClass` and there are the variables `MyClass myclass`, `MyClass myclass20`, `MyClass m` and `YourClass myclass25`. In such a case, IntelliJ IDEA will suggest to rename the first and the second of the variables but won't suggest to rename the third and the fourth.
Rename inheritors	If this option is on, IntelliJ IDEA will look for occurrences of the class or the interface name in the names of its inheritors: – If you are renaming a class, IntelliJ IDEA will search the hierarchies of the classes that extend this class. – If you are renaming an interface, IntelliJ IDEA will search the hierarchies of the interfaces that extend this interface and the hierarchies of the classes that implement this interface.
Rename tests	If this option is on, IntelliJ IDEA will look for occurrences of the class name in the names of the test classes.
Rename bound forms	This option is available if the class you are renaming has an associated GUI form. Select this check box to rename the associated form.

Rename Dialog for a Directory
Refactor | Rename

Shift+F6

Use this dialog to rename a directory.

In addition to renaming the directory itself, IntelliJ IDEA can also look for the usages of the directory name. If found, the changes you are making to the directory name can also be applied to these usages.

The usages are assigned to different categories which correspond to the options which you can turn on and off.

Note that regardless of the options selected, the search scope (that is, the places where the name occurrences are looked for) is always limited to the current entity (file, class, etc) and the entities that the current one depends on.

Item	Description
Rename <directory> and its usages to	Specify a new name for the directory.
Search for references	If this option is on, IntelliJ IDEA will look for occurrences of the directory name in directory references in source code files. Whether or not a string literal is a reference is defined by the (programming) language used in a source file.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the directory name within comments and string literals in your source code files.

Rename Dialog for a Field
Refactor | Rename

Shift+F6

Use this dialog to rename a field.

In addition to renaming the field itself, IntelliJ IDEA can also look for the usages of the field name. If found, the changes you are making to the field name can also be applied to these usages.

The usages are assigned to different categories which correspond to the options which you can turn on and off.

Note that regardless of the options selected, the search scope (that is, the places where the name occurrences are looked for) is always limited to the current entity (file, class, etc) and the entities that the current one depends on.

Item	Description
Rename <field> and its usages to	Specify a new name for the field.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the field name within comments and string literals in your source code files.
Search for text occurrences	If this option is on, IntelliJ IDEA will look for occurrences of the field name in the files that don't contain source code. These may be the text files, properties files, HTML files, documentation files, etc.

Rename Dialog for a File
Refactor | Rename

Shift+F6

Use this dialog to rename a file.

In addition to renaming the file itself, IntelliJ IDEA can also look for the usages of the file name. If found, the changes you are making to the file name can also be applied to these usages.

The usages are assigned to different categories which correspond to the options which you can turn on and off.

Note that regardless of the options selected, the search scope (that is, the places where the name occurrences are looked for) is always limited to the current entity (file, class, etc) and the entities that the current one depends on.

Item	Description
Rename <file> and its usages to	Specify a new name for the file.
Search for references	If this option is on, IntelliJ IDEA will look for occurrences of the file name in file references in source code files. Whether or not a string literal is a reference is defined by the (programming) language used in a source file.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the file name within comments and string literals in your source code files.

Rename Dialog for a Method

Refactor | Rename

Shift+F6

Use this dialog to rename a method.

In addition to renaming the method itself, IntelliJ IDEA can also look for the usages of the method name. If found, the changes you are making to the method name can also be applied to these usages.

The usages are assigned to different categories which correspond to the options which you can turn on and off.

Note that regardless of the options selected, the search scope (that is, the places where the name occurrences are looked for) is always limited to the current entity (file, class, etc) and the entities that the current one depends on.

Item	Description
Rename <method> and its usages to	Specify a new name for the method.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the method name within comments and string literals in your source code files.
Search for text occurrences	If this option is on, IntelliJ IDEA will look for occurrences of the method name in the files that don't contain source code. These may be the text files, properties files, HTML files, documentation files, etc.

Rename Dialog for a Package
Refactor | Rename

Shift+F6

Use this dialog to rename a package.

In addition to renaming the package itself, IntelliJ IDEA can also look for the usages of the package name. If found, the changes you are making to the package name can also be applied to these usages.

The usages are assigned to different categories which correspond to the options which you can turn on and off.

Note that regardless of the options selected, the search scope (that is, the places where the name occurrences are looked for) is always limited to the current entity (file, class, etc) and the entities that the current one depends on.

Item	Description
Rename <package> and its usages to	Specify a new name for the package.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the package name within comments and string literals in your source code files.
Search for text occurrences	If this option is on, IntelliJ IDEA will look for occurrences of the package name in the files that don't contain source code. These may be the text files, properties files, HTML files, documentation files, etc.

Rename Dialog for a Parameter
Refactor | Rename

Shift+F6

Use this dialog to rename a parameter.

In addition to renaming the parameter itself, IntelliJ IDEA can also look for the usages of the parameter name. If found, the changes you are making to the parameter name can also be applied to these usages.

The usages are assigned to different categories which correspond to the options which you can turn on and off.

Note that regardless of the options selected, the search scope (that is, the places where the name occurrences are looked for) is always limited to the current entity (file, class, etc) and the entities that the current one depends on.

Item	Description
Rename <parameter> and its usages to	Specify a new name for the parameter.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the parameter name within comments and string literals in your source code files.

Rename Dialog for a Table or Column

Refactor | Rename

Shift+F6

Use this dialog to rename a table or column.

In addition to renaming the table or column itself, IntelliJ IDEA can also look for the usages of the table or column name. If found, the changes you are making to the table or column name can also be applied to these usages.

Item	Description
Rename <table or column> and its usages to	Specify a new name for the table or column.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the table or column name within comments and string literals in your source code files.
SQL Script	The statement to be run to rename the table or column. If necessary, you can edit the statement right in this pane.
Refactor	Execute the statement and make associated changes right away.
Preview	Preview potential associated changes prior to executing the statement. See Previewing changes .

Rename Dialog for a Variable
Refactor | Rename

Shift+F6

Use this dialog to rename a variable.

In addition to renaming the variable itself, IntelliJ IDEA can also look for the usages of the variable name. If found, the changes you are making to the variable name can also be applied to these usages.

The usages are assigned to different categories which correspond to the options which you can turn on and off.

Note that regardless of the options selected, the search scope (that is, the places where the name occurrences are looked for) is always limited to the current entity (file, class, etc) and the entities that the current one depends on.

Item	Description
Rename <variable> and its usages to	Specify a new name for the variable.
Search in comments and strings	If this check box is selected, IntelliJ IDEA will look for occurrences of the variable name within comments and string literals in your source code files.

Replace Constructor with Builder Dialog

Refactor | Replace Constructor with Builder

The Replace Constructor with Builder refactoring helps hide a constructor, replacing its usages with the references to a newly generated builder class, or to an existing builder class.

Item	Description

Parameters to pass to the builder

Parameter	This column shows the list of parameters detected in the constructor, which will be replaced with the builder fields.
Field name	This editable column shows the list of suggested field names in the builder.
Setter name	This editable column shows the list of suggested setter names in the builder.
Default value	Use this editable column to initialize the fields with the default values.
Optional setter	If the specified default value of a field matches the parameter value in the constructor invocation, then selecting this check box results in omitting setter method for this field in the builder invocation. If this check box is not selected, the corresponding setter method will be shown anyway.

Builder name and location

Create new	Click this radio button to generate a new builder class, with the specified name and destination package.
Builder class name	This editable field shows the suggested name of the new builder class to ge generated. You can accept default, or type a new one.
Package for new builder	Type the name of the destination package, or click the browse button, and locate it in the Choose Destination Package dialog box. If the desired package doesn't exist, click to create a new one.
Use existing	Click this radio button to specify an existing builder class.
Builder class name	Type here the fully qualified name of the desired builder class that already exists in your project, or click the browse button and find it either by name, or in the project tree view.

Replace Constructor with Factory Method Dialog

Refactor | Replace Constructor with Factory Method

Item	Description
Factory method name	Specify here the name for the factory method.
In (fully qualified name)	Specify here the class, where the method will be created.

Replace Inheritance with Delegation Dialog

Refactor | Replace Inheritance with Delegation

Item	Description
Replace with delegation inheritance from	Select here the parent object, inheritance to which will be replaced.
Field name	Specify the name for the field of the new inner class.
Inner class name	In this field specify the name for the inner class definition.
Delegate members	In this area, select the members of the parent class, that will be delegated through the inner class.
Generate getter for delegated component	Check this option to create a getter for the inner class.

Replace Method Code Duplicates Dialog

The Replace Method Code Duplicates allows you to find code repetitions similar to the selected method, and replace them with calls to the method.

Replace Temp with Query Dialog

Refactor | Replace Temp with Query

Item	Description
Name	Specify the name for the extracted method.
Declare static	Check this option to declare the method static. This option is enabled when the initial expression is static itself.
Parameters	In this area, select the parameters you want to be used in the extracted method. The parameters are all checked by default. If unchecked, the appropriate value will be used as a local variable in the extracted method.
Move Up/Down	Use this buttons to reorder parameters in the list.
Visibility	Here you can change the method's visibility scope.

Safe Delete Dialog

Refactor | Safe Delete

`Alt+Delete`

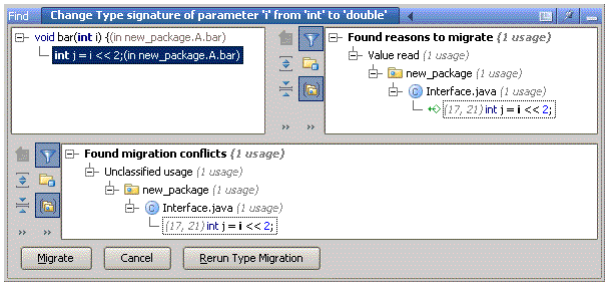
Use this dialog to define the scope of the Safe Delete refactoring.

Item	Description
Search in comments and strings	Select this check box to display the usages of the specified symbol in comments and strings in the Refactoring Preview tool window.
Search for text occurrences	Select this check box to apply the changes text files within the project (such as documentation, HTML, JSP, etc.)

Type Migration Preview

Refactor | Type Migration | Preview

This view allows you to review type migration results and correct the scope of changes by excluding/including items from/to the refactoring.



The view consists of the following panes:

- The upper left pane displays the tree view of the items to be refactored. The root item in this tree view is the one you meant to refactor. Nodes represent all pieces of code that depend on this type.
- The upper right pane is a common Find Usages view.
- The bottom pane shows conflicts found.

Toolbar options for both upper right and bottom panes are similar to those of the Find tool window.

Here context menu options for the upper left pane are described.

Item	Shortcut	Description
Exclude	Delete	Select to exclude the item from the refactoring. It won't be changed.
Include	Insert	Select to include item to the refactoring.
Jump to Source	F4	Opens in the editor the file that contains the selected code fragment, and places the caret at the type.

Use Interface Where Possible Dialog

Refactor | Use Interface Where Possible

Item	Description
Change usages	Select the parent class or interface, which will replace the usages of the current class.
Use interface/superclass in `instanceof`	If this option is checked, IntelliJ IDEA scans the source code for the presence of the `instanceof` statements for the selected superclass or interface, and change it, if found.

Wrap Return Value Dialog

Refactor | Wrap Return Value

Use this refactoring to create a wrapper class around the return values of a method, or use a compatible existing class as a wrapper.

Item	Description
Method to wrap returns from	This read-only field shows the name of the selected method.
Create new class	Click this radio-button to create a new wrapper class. If this option is selected, specify the class and destination package name in the fields below.
Class name	Type the name of the new wrapper class.
Package name	By default, the current package name is displayed. You can type a different package name in the text field, or click the ellipsis button and select the destination package from the tree view. If the desired package doesn't exist, click to create a new one.
Target destination directory	Use this field to select the target destination directory. By default, the current destination directory is displayed. You usually choose the target destination based on a current package. If this package exists in multiple roots, you can click arrow button and select Leave in same source root from the list. In this case, wrapper would be placed near the initial class. You can click the ellipsis button to open Choose Destination Directory window. You can choose Directory Structure tab to select another destination directory or choose Choose By Neighbor Class tab to place a wrapper near the neighbor class if, for example, you want to put the wrapper in `util` directory near your Pair or Triple classes and you do not remember the exact package, putting wrapper near the Pair class would save you time.
Use existing class	Click this radio-button to use an existing class of your choice as a wrapper.
Name	Specify the name of the desired wrapper class. Note that such class should contain a constructor with a parameter of the same type as the return value in question. You can type the fully-qualified class name in the text field, or click the ellipsis button and choose the desired class in the Select parameter class dialog box. Note that you can select the desired wrapper class both from the project and non-project classes.
Wrapper field	Select the field that will store the return value, from the drop-down list of fields, encountered in the specified wrapper class.
Inner class	Click this radio-button to create an inner class. You might want to do that if, for example, you have a private method. In this case you can leave everything in the same class.
Name	Specify the name of the inner class.

Reformat File Dialog

Code | Reformat Code

`Ctrl+Alt+L`

The `Ctrl+Alt+L` command will try to reformat the source code of the specified scope automatically. This dialog box appears when you press `Ctrl+Shift+Alt+L` in the editor of the current file.

Item	Description
Only VCS changed text	If this check box is selected, then reformatting will apply only to the files that have been changed locally , but not yet checked in to the repository. This check box is only available for the files under version control.
Selected text	Choose this option to have the currently selected fragment of source code reformatted.
Whole file	Choose this option to have all the source code in the current file reformatted.
Optimize imports	Select this check box to remove unused import statements from the code within the selected scope.
Rearrange code	Select this check box to reorder your source code entries according to the configurations specified in the Arrangement tab of your Code Style settings . This check box is not available for Python files.
Run	Click this button to start reformatting the source code within the specified scope.

Reformat Code on Directory Dialog

Code | Reformat Code

Ctrl+Alt+L

This dialog box appears when you select a Reformat Code action on the directory.

Item	Description
Directory/Module	This is a read-only field that displays the name of the selected directory or module.
Options	This area displays options for your code reformatting. You can select from the following options: – Include subdirectories: select this check box to have source code from files in the nested directories reformatted. This check box only appears when the nested directories exist. – Optimize imports: select this check box to remove unused import statements from the code within the selected scope. – Rearrange entries: select this check box to reorder your source code entries. – Only VCS changed text: if this check box is selected, then reformatting will apply only to the files that have been changed locally , but not yet checked in to the repository. This check box is only available for the files under version control.
Filters	This area displays options for the filters that you can apply to the code reformatting action. You can select from the following options: – Scope: select this check box to choose, from the drop-down list, a scope to which you want to apply your reformatting options. If you want to configure a custom scope, click ... button to open a Scopes dialog. – File mask(s): select this check box to choose, from the drop-down list, file extensions to which you want to apply your reformatting options.
Run	Click this button to start reformatting the source code within the specified scope.

Rules Alias Definitions Dialog

This dialog appears when you define rules aliases in the Arrangement tab for Java in Code Style dialog.

Item	Description
<alias name>	Use this area to add a new alias definition, check the existing one or remove the ones you do not need. You can use the following commands respectively: – + - click this icon to add a new alias. – — - click this icon to remove an existing alias from the list. – - click this icon to copy a specified rules' sequence to the created alias.
Rules alias definitions	Use this area to define the rules' sequence for the created alias. You can also add a new rule, edit and remove the existing ones, and move them up or down. For more information on the rules, see Code Style reference page, arrangement section.

Register New File Type Association Dialog

The dialog box opens when you attempt to open a file in the editor, but IntelliJ IDEA does not recognize its type by the file extension. Use the dialog box to associate unknown file extensions with existing file types.

Item	Description
Open matching files in IntelliJ IDEA	When this option is selected, IntelliJ IDEA treats the type of the file to be opened as one of the recognized file types. Choose the relevant type from the list box below, that displays all the file types recognized by IntelliJ IDEA.
File Pattern	In this text box, specify the file pattern to be associated with the selected file type. By default, the text box shows the following pattern: `*.<current file full extension>` .
Open matching files in associated application	When this option is selected, IntelliJ IDEA attempts to open the selected file using its native application, if this application is available.

Tip You can change the association later in the File Types settings page.

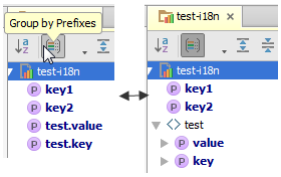
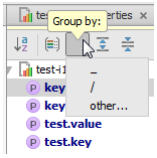
Resource Bundle Editor

View | Jump to Source

Context menu of a resource bundle | Jump to Source

F4

The Resource Bundle Editor is a special tool designed to work with sets of properties files . It shows all files in a bundle, and enables you to perform mass actions on the properties files.

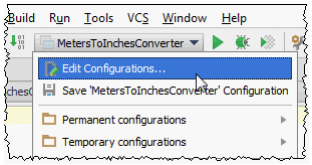
Item	Tooltip and shortcut	Description
	Sort Alphabetically	If this button is pressed, the properties in the left pane are sorted in alphabetical order. Otherwise, they are shown as they are introduced in the properties file.
	Group by Prefixes	If this button is pressed, the properties are grouped according to the separators they have. The Group by drop-down menu enables you to choose the necessary separator:  If the button is not pressed, properties are shown as the items of the property node.
	Group By	Specify the desired separator character, or choose one from the list. 
	Expand all Ctrl+NumPad Plus	Use these buttons to have all nodes expanded or collapsed.
	Collapse all Ctrl+NumPad -	
	F1	Click this button to open the current reference page.

Run/Debug Configurations

Run | Edit Configurations

`Shift+Alt+F10`

`Shift+Alt+F9`



Use this dialog box to create, edit, or remove run/debug configurations , and configure the default settings that will apply to all newly created run/debug configurations.

The default settings are grouped under the Defaults node in the left-hand part of the dialog box.

Click here for the description of the options that are common for all run/debug configurations.

Find the descriptions of specific run/debug configurations in the following pages:

- Run/Debug Configuration: Android Application
- Run/Debug Configuration: Android Test
- Run/Debug Configuration: Applet
- Run/Debug Configuration: Ant Target
- Run/Debug Configuration: Application
- Run/Debug Configuration: Arquillian JUnit
- Run/Debug Configuration: Arquillian TestNG
- Run/Debug Configuration:Behat
- Run/Debug Configuration: Bnd OSGi
- Run/Debug Configuration: CloudBees Deployment
- Run/Debug Configuration: CloudBees Server
- Run/Debug Configuration: Cloud Foundry Deployment
- Run/Debug Configuration: ColdFusion
- Run/Debug Configuration: Compound
- Run/Debug Configuration: Chromium Remote
- Run/Debug Configuration: Cucumber Java
- Run/Debug Configuration: Cucumber.Js
- Run/Debug Configuration: Dart Command Line Application
- Run/Debug Configuration: Dart Test
- Run/Debug Configuration: Docker Deployment
- Run/Debug Configuration: Firefox Remote
- Run/Debug Configuration: Flash App
- Run/Debug Configuration: Flash Remote Debug
- Run/Debug Configuration: FlexUnit
- Run/Debug Configuration: Geronimo Server
- Run/Debug Configuration: GlassFish Server
- Run/Debug Configuration: Google App Engine Deployment
- Run/Debug Configuration: Gradle
- Run/Debug Configuration: Grails
- Run/Debug Configuration. Griffon
- Run/Debug Configuration: Groovy
- Run/Debug Configuration: Grunt.Js
- Run/Debug Configuration: Gulp.Js
- Run/Debug Configuration: GWT
- Run/Debug Configuration: Heroku Deployment
- Run/Debug Configuration: J2ME
- Run/Debug Configuration: Jar Application
- Run/Debug Configuration: Java Scratch
- Run/Debug Configuration: JavaScript Debug
- Run/Debug Configuration: JBoss Server
- Run/Debug Configuration: Jetty Server
- Run/Debug Configuration: JSR45 Compatible Server
- Run/Debug Configuration: JSTestDriver
- Run/Debug Configuration: JUnit
- Run/Debug Configuration: Karma
- Run/Debug Configuration: Kotlin
- Run/Debug Configuration: Kotlin-JavaScript (Experimental)
- Run/Debug Configuration: Kotlin Script

- Run/Debug Configuration: Maven
- Run/Debug Configuration: Meteor
- Run/Debug Configuration: Mocha
- Run/Debug Configuration: MXUnit
- Run/Debug Configuration: Node JS
- Run/Debug Configuration: Node JS Remote Debug
- Run/Debug Configuration: NodeUnit
- Run/Debug Configuration: Node-Webkit
- Run/Debug Configuration: NPM
- Run/Debug Configuration: OpenShift Deployment
- Run/Debug Configuration: OSGi Bundles
- Run/Debug Configuration: PhoneGap/Cordova
- Run/Debug Configuration: PHP Built-in Web Server
- Run/Debug Configuration: PHP HTTP Request
- Run/Debug Configuration: PHP Remote Debug
- Run/Debug Configuration: PHP Script
- Run/Debug Configuration: PHP Web Application
- Run/Debug Configuration: PHPUnit
- Run/Debug Configuration: PHPUnit by HTTP
- Run/Debug Configuration: App Engine for PHP
- Run/Debug Configuration: Plugin
- Run/Debug Configuration: Play2 App
- Run/Debug Configuration: Remote
- Run/Debug Configuration: Resin
- Run/Debug Configuration: SBT Task
- Run/Debug Configuration: Scala
- Run/Debug Configuration: Scala Test
- Run/Debug Configuration: Specs2
- Run/Debug Configuration: Spring Boot
- Run/Debug Configuration: Spring DM Server
- Run/Debug Configuration: Spy-Js
- Run/Debug Configuration: Spy-Js for Node.js
- Run/Debug Configuration: TestNG
- Run/Debug Configuration: Tomcat Server
- Run/Debug Configuration: TomEE Server
- Run/Debug Configuration: Utest
- Run/Debug Configuration: WebLogic Server
- Run/Debug Configuration: WebSphere Server
- Run/Debug Configuration: XSLT

Toolbar

Item	Shortcut	Description
	`Alt+Insert`	Click this button to add a new configuration to the list.
	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of a new run/debug configuration. This field is not available for the default run/debug configurations .
Defaults	This node in the left-hand pane contains the default run/debug configuration settings. Select configuration to modify its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise.

If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Before launch	Use the controls in this area to specify which tasks must performed before applying the run/debug configuration. The tasks are performed in the order they appear in the list. The following options are available: – (Alt+Insert): click this icon to add a task to the list. Select one of the following task types: – Run External tool: select this option to run an application that is external to IntelliJ IDEA. In the dialog that opens , select the application(s) that you want to run. If the required application is not defined in IntelliJ IDEA yet, add its definition (for details see Configuring Third-Party Tools and External Tools). – Run Another Configuration: select this option to execute one on the existing run/debug configuration. Choose a configuration to execute from the dialog box that opens. – Make: select this option to have the project or the module compiled. The Make Module command will be executed if a particular module is specified in the run/debug configuration, or the Make Project command if no modules are specified. – Make, no error check: this option is the same as Make, the only difference being is that IntelliJ IDEA will start the run/debug configuration irrespective of the compilation result. – Build Artifacts: select this option to build artifacts . Select the artifact(s) you want to build in the dialog that opens. For more information, see Working with Artifacts . – Run Ant target: select this option to execute an Ant target. In the dialog that opens select a target that you want to build. For more information, see Ant . – Run Maven goal: select this option to run a Maven goal. In the dialog that opens, select a goal that you want to run. For more information, see Maven . – Run Gradle task: select this option to run a Gradle task. In the dialog that opens, select a task that you want to run. For more information, see Gradle – Compile TypeScript: select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Run File Watcher: select this option to run the File Watchers that are active in the current project. For more information, see Using File Watchers . – Generate CoffeeScript Source Maps: select this option to generate source maps for your CoffeeScript sources. In the dialog that opens, specify the path to the folder where CoffeeScript files will be generated. For more information, see CoffeeScript Support . – Run Remote External tool: select this option to run a remote SSH external tool. In the dialog that opens, specify the SSH external tool that you want to run. For more information, see Remote SSH External Tools . – (Alt+Delete): click this icon to remove the selected task from the list. – (Enter): click this icon to modify the selected task. Edit the task settings in the dialog that opens. – (Alt+Up): click this icon to move the selected task up in the list. – (Alt+Down): click this icon to move the selected task down in the list. – Show this page: select this check box if you want to display the run/debug configuration settings before applying it. – Active tool window: select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Item **Description**

Confirm rerun with process termination	The behavior of this button depends on selecting the check box Single instance only for a particular run/debug configuration. – If this check box is selected, then, in case of a single instance, launching a new process (for example, by clicking ▶ on the main toolbar) while the other process is still running, results in showing a dialog box, where one should confirm termination of the current process and launching a new one. – If this check box is not selected (or in case of multiple instances), IntelliJ IDEA starts the new process silently.
Temporary configurations limit	Specify here the maximum number of temporary configurations to be stored and shown in the Select Run/Debug Configuration drop-down list.

Run/Debug Configuration: Android Application

Use this dialog box to create configurations used to run or debug Android applications and activities on actual or virtual devices.

This dialog box consists of three tabs:

- General
- Miscellaneous
- Profiling

The dialog also provides a toolbar and the options that are common to all run/debug configuration types.

General Tab

Item	Description
Module	From this drop-down list, select a module to which this configuration will be applied.

Installation Options	Use this area to configure the following settings:
	– Deploy - use this drop-down list to appoint the `.apk` file that will be deployed to the target virtual or physical device. You can select one of the following options:
	  – Default APK - select this option to have IntelliJ IDEA upload the `.apk` built from the module specified in the Module drop-down list above. The `.apk` is built automatically, no preliminary artifact configuration is required from your side.
	  – Custom Artifact - select this option to have IntelliJ IDEA upload the `.apk` built from the code and resources appointed in a manually configured artifact. Select an artifact from the drop-down list which shows all manually created artifact definitions for the selected module. See Working with Artifacts for details.
	  – Nothing - select this option to suppress uploading data to the device. This approach may be helpful if you have already deployed your application manually outside IntelliJ IDEA.
	– Install Flags - use this field to specify options for the `pm install` command.
	
	For more information, please see package manager page.

Launch Options	In this area, appoint the activity that will be launched on the application start.
	– Launch - use this drop-down list to specify the following launch settings:
	  – Default Activity - select this option to have IntelliJ IDEA automatically launch the activity marked as start-up .
	  – Nothing - select this option to connect to an already running application provided that you have chosen Nothing in the Installation Options area. Executing a run configuration with these settings is the same as clicking the Attach debugger to Android process button on the toolbar (see Debugging a Running Application).
	  – Specified Activity - select this option to have IntelliJ IDEA launch an activity of your choice upon the application start. This option may be helpful when you want to run or debug a part of your application and have already chosen the Custom Artifact option in the Installation Options area. Type the activity name manually or click the Browse button and select it in the Select Activity Class dialog box that opens.
	The list of available activities is determined by the choice of the module.
	  – URL - select this option to launch a browser when you run your application. You can specify the URL address in the URL field.
	– Launch Flags - use this field to specify additional options for the `am start` command.
	
	For mor information, please see activity manager page.

Deployment Target Options	In this area, select the type of device to run/debug the application on. The available options for Target are: – Show Device Chooser Dialog: select this option to have IntelliJ IDEA display the Choose Device dialog box when the application is started. If you want to automatically use the device chosen through the Choose Device dialog in the future, select the Use same device for future launches option. – USB Device: select this option to have IntelliJ IDEA detect a plugged-in USB device upon the application start. Note Selecting the Show Device Chooser Dialog or USB Device option may be helpful if you are going to run the application on a physical device which will be plugged in later and therefore the set of available devices cannot be foreseen. – Emulator: select this option to use one of the configured device emulators. From the Prefer Android Virtual Device drop-down list, select a virtual device that will be used to run/debug the specified activity. If you want to add/edit a virtual device configuration, click the Browse button to launch the Android Virtual Device (AVD) Manager (see Managing Virtual Devices for details).

Profiling

In this tab, specify Graphic Trace Options.

Item	Description
Disable precompiled shaders and programs	Select this check box if you want to disable precompiled shaders and programs and compile them at the runtime instead.

Miscellaneous

In this tab, configure the scope of log data shown during a run/debug session.

Item	Description
Logcast	Use this area to configure logcast settings for your Android run/debug configuration. You can select from the following options: – Show logcast automatically - select this check box to show the logcast automatically every time an application is deployed and launched successfully. – Clear log before launch - select this check box if you want data from previous sessions to be removed from the log file before starting the application.
Installation Options	Use this area to configure installation options for running or debugging your Android application . You can select the following options: – Skip installation if APK has not changed - select this check box to skip APK installation when you run your application in case APK was not changed. – Force stop running application before launching activity - select this check box to stop running the already launched application before you start running the application from IntelliJ IDEA.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of a new run/debug configuration. This field is not available for the default run/debug configurations .
Defaults	This node in the left-hand pane contains the default run/debug configuration settings. Select configuration to modify its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Before launch	Use the controls in this area to specify which tasks must performed before applying the run/debug configuration. The tasks are performed in the order they appear in the list. The following options are available: − + (Alt+Insert): click this icon to add a task to the list. Select one of the following task types: − Run External tool: select this option to run an application that is external to IntelliJ IDEA. In the dialog that opens , select the application(s) that you want to run. If the required application is not defined in IntelliJ IDEA yet, add its definition (for details see Configuring Third-Party Tools and External Tools). − Run Another Configuration: select this option to execute one on the existing run/debug configuration. Choose a configuration to execute from the dialog box that opens. − Make: select this option to have the project or the module compiled. The Make Module command will be executed if a particular module is specified in the run/debug configuration, or the Make Project command if no modules are specified. − Make, no error check: this option is the same as Make, the only difference being is that IntelliJ IDEA will start the run/debug configuration irrespective of the compilation result. − Build Artifacts: select this option to build artifacts . Select the artifact(s) you want to build in the dialog that opens. For more information, see Working with Artifacts . − Run Ant target: select this option to execute an Ant target. In the dialog that opens select a target that you want to build. For more information, see Ant . − Run Maven goal: select this option to run a Maven goal. In the dialog that opens, select a goal that you want to run. For more information, see Maven . − Run Gradle task: select this option to run a Gradle task. In the dialog that opens, select a task that you want to run. For more information, see Gradle − Compile TypeScript: select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: − If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. − If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. − Run File Watcher: select this option to run the File Watchers that are active in the current project. For more information, see Using File Watchers . − Generate CoffeeScript Source Maps: select this option to generate source maps for your CoffeeScript sources. In the dialog that opens, specify the path to the folder where CoffeeScript files will be generated. For more information, see CoffeeScript Support . − Run Remote External tool: select this option to run a remote SSH external tool. In the dialog that opens, specify the SSH external tool that you want to run. For more information, see Remote SSH External Tools . − − (Alt+Delete): click this icon to remove the selected task from the list. − (Enter): click this icon to modify the selected task. Edit the task settings in the dialog that opens. − ↑ (Alt+Up): click this icon to move the selected task up in the list. − ↓ (Alt+Down): click this icon to move the selected task down in the list. − Show this page: select this check box if you want to display the run/debug configuration settings before applying it. − Active tool window: select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Android Test

Use this dialog box to configure testing of Android applications and activities on actual or virtual devices.

The dialog box consists of three tabs:

- General
- Emulator
- Logcat

The dialog also provides a toolbar and the options that are common to all run/debug configuration types.

General Tab

Item	Description
Module	From this drop-down list, select the module to which this configuration will be applied.
Test	In this area, specify the location of tests that you want to apply: – All in module: select this option if you want to launch all tests from the selected module. – All in package: select this option is you want to launch all tests from a specific package. In the Package text box, specify the package where tests are located. Type the package name manually or click the Browse button and select the desired package in the dialog that opens. – Class: select this option if you want to launch tests of a specific class. In the Class text box, specify the required class. Type the class name manually or click the Browse button and select the desired class in the dialog that opens. – Method: select this option if you want to launch a specific test method. In the Class text box, specify the class to which this method belongs. In the Method text box, specify the desired method. Type the names manually or click the Browse button and select the desired class and method in the dialog that opens.
Specific instrumentation runner	In this text box, specify the location of the desired instrumentation runner. Type the path manually or click the Browse button and select the location in the dialog that opens.
Deployment Target Options	In this area, select the type of device to run/debug the application on. The available options for Target are: – Show Device Chooser Dialog: select this option to have IntelliJ IDEA display the Choose Device dialog box when the application is started. If you want to automatically use the device chosen through the Choose Device dialog in the future, select the Use same device for future launches option. – USB Device: select this option to have IntelliJ IDEA detect a plugged-in USB device upon the application start. **Note** Selecting the Show Device Chooser Dialog or USB Device option may be helpful if you are going to run the application on a physical device which will be plugged in later and therefore the set of available devices cannot be foreseen. – Emulator: select this option to use one of the configured device emulators. From the Prefer Android Virtual Device drop-down list, select a virtual device that will be used to run/debug the specified activity. If you want to add/edit a virtual device configuration, click the Browse button to launch the Android Virtual Device (AVD) Manager (see Managing Virtual Devices for details).

Emulator Tab

In this tab, specify additional configuration settings for launching the Android emulator.

Item	Description
Network Speed	From this drop-down list, select the network transfer rate to be emulated.
Network Latency	Use this drop-down list to specify the time delay between the initial input and the output. You can select between the following latency levels: – None: no latency – GPRS: GPRS (min 150, max 550 milliseconds) – EDGE: EDGE/EGPRS (min 80, max 400 milliseconds) – UMTS: UMTS/3G (min 35, max 200 milliseconds)
Wipe user data	Select this check box to have the contents of the user-data image copied to the new user-data disk image while resetting the image. By default, the `<system>/userdata.img` is copied.
Disable boot animation	Select this check box if you do not want an animated boot screen displayed upon the emulator start-up.
Additional command line options	In this text box, type the additional options to be passed to the emulator via the command line. For lengthy command lines, click the button and type the text in the Emulator Additional Command Line Options dialog that opens.

Logcat Tab

Item	Description
Clear log before launch	Select this check box if you want data from previous sessions to be removed from the log before starting the application.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.

	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of a new run/debug configuration. This field is not available for the default run/debug configurations .
Defaults	This node in the left-hand pane contains the default run/debug configuration settings. Select configuration to modify its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Before launch	Use the controls in this area to specify which tasks must performed before applying the run/debug configuration. The tasks are performed in the order they appear in the list. The following options are available: – (`Alt+Insert`): click this icon to add a task to the list. Select one of the following task types: – Run External tool: select this option to run an application that is external to IntelliJ IDEA. In the dialog that opens , select the application(s) that you want to run. If the required application is not defined in IntelliJ IDEA yet, add its definition (for details see Configuring Third-Party Tools and External Tools). – Run Another Configuration: select this option to execute one on the existing run/debug configuration. Choose a configuration to execute from the dialog box that opens. – Make: select this option to have the project or the module compiled. The Make Module command will be executed if a particular module is specified in the run/debug configuration, or the Make Project command if no modules are specified. – Make, no error check: this option is the same as Make, the only difference being is that IntelliJ IDEA will start the run/debug configuration irrespective of the compilation result. – Build Artifacts: select this option to build artifacts . Select the artifact(s) you want to build in the dialog that opens. For more information, see Working with Artifacts . – Run Ant target: select this option to execute an Ant target. In the dialog that opens select a target that you want to build. For more information, see Ant . – Run Maven goal: select this option to run a Maven goal. In the dialog that opens, select a goal that you want to run. For more information, see Maven . – Run Gradle task: select this option to run a Gradle task. In the dialog that opens, select a task that you want to run. For more information, see Gradle – Compile TypeScript: select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Run File Watcher: select this option to run the File Watchers that are active in the current project. For more information, see Using File Watchers . – Generate CoffeeScript Source Maps: select this option to generate source maps for your CoffeeScript sources. In the dialog that opens, specify the path to the folder where CoffeeScript files will be generated. For more information, see CoffeeScript Support . – Run Remote External tool: select this option to run a remote SSH external tool. In the dialog that opens, specify the SSH external tool that you want to run. For more information, see Remote SSH External Tools . – (`Alt+Delete`): click this icon to remove the selected task from the list. – (`Enter`): click this icon to modify the selected task. Edit the task settings in the dialog that opens. – (`Alt+Up`): click this icon to move the selected task up in the list. – (`Alt+Down`): click this icon to move the selected task down in the list. – Show this page: select this check box if you want to display the run/debug configuration settings before applying it. – Active tool window: select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Applet

An applet run/debug configuration enables you to run Java applets using an applet viewer or a Web page and your local browser.

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Item	Description
Name	In this text box, specify the name of the current run/debug configuration.
Applet Class	Choose this option to create a configuration profile for running an applet via an applet class.
URL	Choose this option to create a configuration profile for running an applet via a Web page.
Applet Class	Use the controls in the area to specify the applet implementation class and the applet size. – Applet Class - in this text box, specify the fully qualified name of the applet class to run. Type the name manually or click the Browse button to open the Choose Applet Class dialog box and select the path there. – Width - in this text box, type the desired width of the applet. – Height - in this text box, type the desired height of the applet. The area is available after the Applet Class option is chosen.
Applet Parameters	In this area, specify a set of applet parameters. – Add - click this button to add a new entry to the list. – Remove - click this button to delete the selected entry from the list. – Name - in this text bos, specify the name of the applet parameter. – Value - in this text bos, specify the parameter value. This area is available after the Applet Class option is chosen.
URL	In the URL/HTML File text box, specify the file to invoke the applet from. Type the path to the file manually or click the Browse button and select the file in the dialog that opens . This area is available after the URL option is chosen.
Policy File	In this text box, specify the location of tje `appletviewer.policy` file. Because this file is distributed with IntelliJ IDEA, the text box is by default filled in with the standard location. To specify another location, click the Browse button and choose the desired location in the dialog that opens .
VM Options for Applet Viewer	In this text box, specify the string to be passed to the VM for executing the applet viewer. Usually this string contains the options such as `-mx` , `-verbose` , etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . The `-classpath` option specified in this field overrides the classpath of the module.
Use Classpath and JDK of the Module	From this drop-down list, select one of the modules configured in your project. The classpath and JDK of this module will be used to run the applet with the current run configuration.
Use Alternative JRE	Select this check box to enable defining another JRE than the JRE used by the current project / module.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise.

If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Ant Target

This dialog lets you create a run/debug configuration for the selected Ant target.

Item	Description
Name	This field shows the name of your Ant target that you have selected in the Ant Build tool window.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Application

The application run/debug configuration enables you running or debugging applications via the `main()` method .

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

Click here for the description of the options that are common for all run/debug configurations.

Configuration tab

Item	Description
Main class	In this text box, specify the fully qualified name of the class to be executed (passed to the JRE). Type the class name manually or click the Browse button to open the Choose Main Class dialog box, where you can find the desired class by name or search through the project.
VM options	In this text box, specify the string to be passed to the VM for launching an application. Usually this string contains the options such as `-mx` , `-verbose` , etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options .
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens . Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. Note The list of environment variables may vary depending on the enabled plugins.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values. Note that you can copy-paste the contents of the Environment variables field without having to open the Environment Variables dialog box.
Use classpath of module	Select the module whose classpath should be used to run the application.
JRE	By default, the newest JDK from the module dependencies is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Enable capturing form snapshots	Select this check box to enable the GUI Designer to take snapshots of the GUI components , that can be afterwards converted into a form.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool

window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑ ↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.

See also, Working with Artifacts .

- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Edit Log Files Aliases Dialog

The dialog opens when you click the + or the buttons in the Logs tab of the Run/Debug Configuration dialogs .

Use this dialog to specify a file or a group of files that you want to be displayed on dedicated tabs of the Run or Debug tool window .

Item	Description
Alias	In this text box, type the alias for the log entry. This alias will be displayed in the Logs tab of the Run/Debug Configuration: Application dialog box and in the header of the dedicated tab of the Run or Debug tool window.
Log File Location	In this text box, specify the log files to display during running or debugging. Do one of the following: – Specify the full path to a specific file. Type the path manually or click the Browse button and choose the file in the dialog that opens . – Specify the base directory and add an Ant pattern that defines the fileset to be displayed.
Show All Files Coverable by Pattern	Select this check box to have IntelliJ IDEA open a separate dedicated tab for each log file that matches the specified pattern.

Please note the following:

– If no alias is specified, the dedicated tab header shows the path to each log file.

– If a pattern covers more than one file, the tab header shows the name of the file instead of the log entry alias, even is an alias is specified.

Run/Debug Configuration: Arquillian JUnit

Run | Edit Configurations | + | Arquillian JUnit

Arquillian JUnit run/debug configurations let you run and debug your Arquillian JUnit tests. (The JBoss Arquillian Support plugin must be enabled.)

- Name, Share and Single instance only
- Arquillian Container tab
- Configuration tab
- Code Coverage tab
- Logs tab
- Before Launch options
- Toolbar

See also, Arquillian: a Quick Start Guide .

Name, Share and Single instance only

Item	Description
Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.

Arquillian Container tab

To configure your Arquillian containers, click Configure. Then select the container to be used.

See also, Arquillian Containers .

Configuration tab

This tab lets you specify the settings for JUnit .

Item	Description
Test kind	From this drop-down list, select the scope for your tests and fill in the fields depending on your selection.
All in package	Select this option to run all unit tests in the specified package. Fill in the following fields:
Package	Specify package name
Search for tests	Select where in your project IntelliJ IDEA shall look for test classes related to the current package: – In whole project: IntelliJ IDEA will look for test classes in all project modules – In single module: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field – Across multiple dependencies: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field, and in the modules that depend on it
All in directory	Select this option to run all unit tests in the specified directory. Fill in the following field:
Directory	Specify the directory where you want to run the tests. It will act as the root directory for all relative input and output paths.
Pattern	Select this option to run a set of test classes. This set may include classes located in the same or different directories, packages or modules. Fill in the following fields:
Pattern	Specify the required classes. Each class in this field must be represented by its fully qualified name. Class names must be separated with `||`. You can type class names manually, or click + on the right (or press `Shift+Enter`) and search for classes you want to add in the dialog that opens. You can also create a suite test, i.e. a bundle of several test classes that will be run together. To create a suite test class, click on the right and type the test classes you want to be run as a suite in the Configure suit tests dialog that opens. As a result, a new class will be created with the `@Suite` annotation.
Method	Specify the method to be launched (passed to the JRE). Type method name, or click ... and select the desired method in the dialog that opens.
Search for tests	Select where in your project IntelliJ IDEA shall look for test classes related to the current package: – In whole project: IntelliJ IDEA will look for test classes in all project modules – In single module: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field – Across multiple dependencies: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field, and in the modules that depend on it
Class	Select this option to run all tests in a class. Fill in the following field:
Class	Specify the fully qualified name of the class to be launched (passed to the JRE). Type the class name or click ... and select the desired class in the dialog that opens.
Method	Select this option to run an individual test method. Fill in the following fields:
Class	Specify the fully qualified name of the class to be launched (passed to the JRE). Type the class name or click ... and select the desired class in the dialog that opens.
Method	Specify the method to be launched (passed to the JRE). Type method name, or click ... and select the desired method in the dialog that opens.
Category	Select this option if you only want to run test classes and test methods that are annotated either with the category given with the `@IncludeCategory` annotation, or a subtype of this category. Learn more about JUnit categories .

Fill in the following fields:

Category	Specify the desired category. Type category name, or click ... and select the desired category in the dialog that opens.
Search for tests	Select where in your project IntelliJ IDEA shall look for test classes related to the current package: – In whole project: IntelliJ IDEA will look for test classes in all project modules – In single module: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field – Across multiple dependencies: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field, and in the modules that depend on it
Fork mode	This option controls how many Java VMs will be created if you want to fork some tests. Select method or class to create a separate virtual machine for each method or class respectively. The available options in this drop-down list depend on the Test kind setting.
Repeat	If you want to repeatedly run a test, select the threshold from this drop-down list. You can select to run your test once, n times (in this case specify the number of times in the field on the right), until the test fails, or until it is stopped.
VM options	If necessary, specify the string to be passed to the VM. This string may contain the options such as `-mx`, `-verbose`, etc. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. If there is not enough space, you can click and enter the string in the dialog that opens. The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this field, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the required arguments in the dialog that opens. Use the same rules as for specifying the VM options.
Working directory	Specify the directory that will act as the current directory when running the test. It will act as the root directory for all relative input and output paths. By default, the directory where the project file resides, is used as a working directory. Type directory name, or click ... and select the desired directory in the dialog that opens. You can also click to switch between directories.
Environment variables	Click ... to open the Environment Variables dialog box where you can create variables and specify their values.
Use classpath of module	Select the module whose classpath should be used to run the tests.
JRE	Specify the JRE to be used. Select the JRE from the list, or click ... and select the installation folder of the required JRE in the dialog that opens.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results.
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window.

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window.
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed.

	– Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click –. See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: Arquillian TestNG

Run | Edit Configurations | + | Arquillian TestNG

Arquillian TestNG run/debug configurations let you run and debug your Arquillian TestNG tests. (The JBoss Arquillian Support plugin must be enabled.)

- Name, Share and Single instance only
- Arquillian Container tab
- Configuraion tab
- Code Coverage tab
- Logs tab
- Before Launch options
- Toolbar

See also, Arquillian: a Quick Start Guide .

Name, Share and Single instance only

Item	Description
Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.

Arquillian Container tab

To configure your Arquillian containers, click Configure. Then select the container to be used.

See also, Arquillian Containers .

Configuraion tab

This tab lets you specify the settings for TestNG .

Item	Description
All in package	Run all tests in a package. Package. The fully qualified name of the package. In whole project. IntelliJ IDEA will look for the tests in all the modules. In single module. IntelliJ IDEA will look for the tests only in the module that is selected in the Use classpath of module field . Across module dependencies. The same as the previous option plus the modules that depend on that module.
Suite	Run a test suite. Suite. Specify the corresponding `testng.xml` file.
Group	Run a test group. Group. The group to be run. Learn more about TestNG groups .
Class	Run a test class. Class. The fully qualified name of the test class to be run.
Method	Run a test method. Class. The fully qualified name of the test class. Method. The name of the method to be run.
Pattern	Run the tests that conform to the specified pattern. Pattern. Form the pattern by clicking + and then selecting one or more TestNG test classes. Alternatively, click and type the pattern in the dialog that opens.
Output directory	The directory in which test reports will be generated.
JDK Settings	
VM options	Options and arguments to be passed to the JVM in which the tests run. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . The `-classpath` option specified in this field overrides the classpath of the module.
Test runner parameters	Arguments to be passed to the test runner. Use the same rules as for specifying the VM options .
Working directory	The current working directory for the tests.

Environment variables	The environment variables to be passed to the corresponding JVM.
Use classpath of module	The module whose classpath is used when running the tests.
JRE	The JRE to be used.
Parameters	
Properties file	Specify the `.properties` file to be passed to TestNG.
Name - Value	Additional paramenters as key - value pairs.
Listeners	
+ −	Use these icons to make up a list of listeners.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
−	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see

Configuring Third-Party Tools and External Tools .

- Make. Select this option to compile the project.
  If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration.
- Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result.
- Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run.
- Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant .
- Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located.
  For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .

	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
	Alt+Insert	Create a run/debug configuration.
	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click . See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration:Behat

This dialog box is available only when the PHP and Behat plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Use this dialog box to create a configuration to be used for running and debugging unit tests on PHP applications using the Behat framework.

On this page:

– Before You Start
– Test Runner Area
– Command Line Area
– Toolbar
– Common options

Before You Start

To run Behat tests:

1. Install and configure the Behat framework on your computer as described in Using Behat Framework .
2. Make sure the PHP and Behat plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Test Runner Area

In this area, specify the scenarios to launch and the command line switches to be passed to Behat.

Item	Description
Test scope	In this area, specify the location of scenarios or the configuration file where they are listed. – Directory: select this option to have all the scenarios in a directory launched. In the Directory text box, specify the directory to search for `.feature` files with scenarios in. Type the path to the directory manually or click the Browse button … and select the desired directory in the Choose Test Directory dialog box, that opens. – File: select this option to have all the scenarios in a specific `.feature` file launched. 1. In the File text box, specify the `.feature` file to search the scenarios in. Type the path to the file manually or click the Browse button … and select the desired directory in the dialog box, that opens. 2. In the Class text box, specify the desired class. Type the class name manually or click the Browse button … and select the desired class in the tree view, that opens. – Scenario: select this option to have a specific scenario launched. 1. In the File text box, specify the `.feature` file to search for the scenario in. Type the file name manually or click the Browse button … and select the desired file in the tree view, that opens. 2. In the Scenario text box, specify the desired scenario. – Defined in the configuration file: select this option to have Behat execute the tests from a dedicated `.yaml` configuration file. By default, Behat uses the configuration file appointed in the Test Runner area of the Behat . In its turn, this can be either the `behat.yaml` file, for which Behat searches in the project root or in the `config` folder, or any other `.yaml` configuration file which you specified as Default during the initial configuration of Behat in IntelliJ IDEA. – To have the default for all Behat run configurations file used, clear the Use alternative configuration file check box. – To launch scenarios from a custom configuration file, select the Use alternative configuration file check box and specify the location of the desired `.yaml` file in the text box next to it. – To open the Behat page and specify another default configuration file to use, click the button.

Command Line Area

In this area, customize the behavior of the current PHP interpreter by specifying the options and arguments to be passed to the PHP executable file.

Item	Description
Interpreter options	In this text box, specify the options to be passed to the PHP executable file. They override the default behavior of the PHP interpreter and/or ensure that additional activities are performed. If necessary, click and type the desired options in the Command Line Options dialog box. Type each option on a new line. When you close the dialog box, they are all displayed in the Command line options text box with spaces as separators.
Custom working directory	In this text box, specify the location of the files that are outside the folder with tests and are referenced in your tests through relative paths. Type the path manually or click the Browse button … and select the desired folder in the dialog that opens . This settings does not block the test execution because the location of tests is always specified through a full path to the corresponding files and/or directories.
Environment variables	In this field, specify the environment variables be passed to the built-in server. See Environment Variables in Apache for details.

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
–	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.

	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ⬆⬇ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
⬆	Alt+Up	Click this icon to move the selected task one line up in the list.
⬇	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Bnd OSGI

In this part:

- Run Launcher
- Test Launcher (JUnit)

Run Launcher

Item	Description
Bnd run descriptor	Use this field to select bnd run descriptor that is represented in a form of either `*.bnd` or `*.bndrun` file. The descriptor contains the description of bundles, its options and the execution.
JRE	By default, the project JDK is used to run the bundles. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog

that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Test Launcher (JUnit)

Item	Description
Bnd run descriptor	Use this field to select bnd run descriptor for JUnit tests that is represented in a form of either `*.bnd` or `*.bndrun` file. The descriptor contains the description of bundles, its options and the execution.
JRE	By default, the project JDK is used to run the bundles. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog

that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: CloudBees Deployment

Run | Edit Configurations | + | CloudBees Deployment

CloudBees Deployment run/debug configurations let you deploy your application artifacts to CloudBees . (The CloudBees integration plugin must be enabled.)

– Main settings

– Before Launch options

– Toolbar

Main settings

Item	Description
Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.
Server	Select the cloud access configuration to be used. To create a new configuration, or to edit an existing one, click … (Shift+Enter). For more information, see CloudBees.
Deployment	Select the application artifact to be deployed. Only archive artifact formats can be used (e.g. WAR, EAR).
ClickStack	Select the target application container (ClickStack) for your application.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Debug tool window opens when you start the run/debug configuration in the debug mode. Otherwise, the tool window isn't shown. However, when the configuration is running in the debug mode, you can open the Debug tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder.

To remove grouping, select a folder and click —.

See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: CloudBees Server

Run | Edit Configurations | + | CloudBees Server

CloudBees Server run/debug configurations let you deploy and debug applications intended for CloudBees locally. (The CloudBees integration plugin must be enabled.)

The Tomcat instance embedded in CloudBees is used as a server. This Tomcat instance is included in the server client libraries. You can download those libraries using a quick fix provided right in the run configuration:

If the message *Error: Client libraries were not downloaded* is shown in the lower part of the dialog, click Fix.

- Name field and Share option
- Server tab
- Deployment tab
- Logs tab
- Code Coverage tab
- Before Launch options
- Toolbar

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab

Item	Description
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. Most likely, the default `http://localhost:8080` will do.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
HTTP port	The server HTTP port.
JNDI port	The server JNDI port.

Deployment tab

Specify an artifact or an external resource to be deployed on the server. (An external resource means a deployable Web component such as a WAR file which is not represented by a project artifact.)

There should be exactly one item in the deployment list. "Exploded" artifacts cannot be used.

To add an item to the list, click + and do one of the following:

– To add an artifact, select Artifact and choose the desired artifact in the dialog that opens.
– To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .

– (Alt+Delete) removes the selected items from the list.

(F4) opens the Artifacts page of the Project Structure dialog for the selected artifact.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use or to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click –.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven .

– Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .

	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
	Alt+Insert	Create a run/debug configuration.
	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click . See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: Cloud Foundry Deployment

Run | Edit Configurations | + | CloudFoundry Deployment

Cloud Foundry Deployment run/debug configurations let you deploy your application artifacts to Cloud Foundry . (The Cloud Foundry integration plugin must be enabled.)

– Main settings
– Before Launch options
– Toolbar

Main settings

Item	Description
Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.
Server	Select the cloud access configuration to be used. To create a new configuration, or to edit an existing one, click ... (Shift+Enter). For more information, see Cloud Foundry.
Deployment	Select the application artifact to be deployed. Only archive artifact formats can be used (e.g. WAR, EAR).
Use custom domain	If you want to use a custom domain, select the check box and specify the domain in the field. Otherwise, the default shared domain is used.
Memory	The memory (RAM) limit per one application instance. <Default> corresponds to the default quota plan.
Instances	The number of application instances to be started. If nothing is specified, one instance is assumed.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Debug tool window opens when you start the run/debug configuration in the debug mode. Otherwise, the tool window isn't shown. However, when the configuration is running in the debug mode, you can open the Debug tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up /	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main

Alt+Down toolbar.

You can group run/debug configurations by placing them into folders.
To create a folder, select the configurations to be grouped and click . Specify the name of the folder.

Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder.

To remove grouping, select a folder and click .

See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: ColdFusion

A ColdFusion run/debug configuration enables you to deploy and run applications on the ColdFusion server.

Click here for the description of the options that are common for all run/debug configurations.

Item	Description
Web Path	In this text box, specify the URL address that corresponds to the server root. The server side of deployment mappings will be specified relative to this URL.
Browser	In this drop-down list, specify the Web browser to open the application in. The list shows all the Web browsers configured in the IDE .

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are

detected:

- If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
- If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.

- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Compound

Use this dialog box to create configurations containing multiple run/debug configurations that you can launch at once. This is useful, for example, if you want to launch various automated tests and get test results in one window.

Press + to select which of the existing configurations you want to include into the Compound configuration, and fill in the following fields:

Item	Description
Name	Specify the name of the new Run/Debug Configuration.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Run/Debug Configuration: Chromium Remote

Use this dialog box to create configurations for debugging applications remotely in ChromiumOS .

On this page:

– Chromium Remote-Specific Configuration Settings
– Toolbar
– Common options

Chromium Remote-Specific Configuration Settings

Item	Description
Host	In this text box, specify the host where the application is running.
Port	In this spin box, specify the port to connect to.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the

		`package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Cucumber Java

Cucumber run/debug configuration enables you to run features or scenarios via the `cucumber.cli.main` class.

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

Configuration tab

Item	Description
Main class	In this text box, specify the fully qualified name of the class with the `main()` method. This class is taken from the jar archive attached when enabling Cucumber support in project. By default, the main class name is `cucumber.cli.Main`. Type the class name manually or click the Browse button to open the Choose Main Class dialog box, where you can find the desired class by name or search through the project.
Glue	In this text field, specify the name of the package, where step definitions are stored.
VM options	In this text box, specify the string to be passed to the VM for launching an application. Usually this string contains the options such as `-mx`, `-verbose`, etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options .
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens . Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. Note The list of environment variables may vary depending on the enabled plugins.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values. Note that you can copy-paste the contents of the Environment variables field without having to open the Environment Variables dialog box.
Use classpath of module	Select the module whose classpath should be used to run the application.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a

particular module is specified in the run/debug configuration, and the Make Project command otherwise.
If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration.

- Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result.
- Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Cucumber.js

In this dialog box, create configurations for running and debugging JavaScript unit tests using the Cucumber.js test runner.

On this page:

- Getting Access to the Run/Debug Configuration: Cucumber.js Dialog Box
- Cucumber.js-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: Cucumber.js Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS runtime environment that contains the Node Package Manager(npm) .
3. Using the Node Package Manager, install the Cucumber.js as described in Preparing to Use Cucumber.Js Test Runner .
4. Install and enable the Cucumber.js and Gherkin plugins. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Cucumber.js-Specific Configuration Settings

Item	Description
Feature file or directory	In this text box, specify the tests to run. Cucumber.js runs tests that are called features and are written in the Gherkin language. Each feature is described in a separate file with the extension `feature`. – To have one feature executed, specify the path to the corresponding `.feature` file. – To have a bunch of features executed, specify the folder where the `.feature` files to run are stored. Type the path manually or click the Browse button … and choose the file or folder in the dialog box that opens.
Cucumber.js arguments	In this text box, specify the command line arguments to be passed to the Cucumber.js executable file, such as `-r` (`--require LIBRARY|DIR`), `-t` (`--tags TAG_EXPRESSION`), or `--coffee` . For details, see Cucumber's native built-in help available through the following command: `cucumber-js --help`
Executable path	In this text box, specify the location of the cucumber executable file, `.cmd` , `.bat` , or other depending on the operating system used. The location depends on the installation mode, see Installing the Cucumber.js test runner . Type the path manually or click the Browse button … and choose the file in the dialog box that opens.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a

confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Dart Command Line Application

In this dialog box, create a configuration to be used for running and debugging the client-side Dart code.

On this page:

- Getting Access to the Run/Debug Configuration: Dart Command Line Application Dialog Box
- Dart Command Line Application-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: Dart Command Line Application Dialog Box

1. Make sure the Node.js and Dart plugins are activated. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Dart Command Line Application-Specific Configuration Settings

Item	Description
Name	In this text box, specify the name of the run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Dart file	In this text box, specify the Dart file to start the application from. The file must contain a `main()` method. Type the path to the file manually or click the Browse button and select the file in the dialog box that opens.
VM options	In this text box, specify the options to launch the Dart Virtual Machine with. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Program arguments	In this text box, type the list of arguments to start the application with. Use the same format as for VM options.
Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button . – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.
Working Directory	In this field, specify the working directory of the application. All references in the starting NodeJS application file, for example, includes, will be resolved relative to this folder, unless such references use full paths. By default, the field shows the project root folder. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button and select the location in the dialog box, that opens.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Dart Test

In this dialog box, create configurations to run unit tests for Dart applications.

On this page:

- Getting Access to the Run/Debug Configuration: Dart Test Dialog Box
- Dart Test Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: Dart Test Dialog Box

1. Make sure the Dart development environment is configured, see Preparing for Dart Development .
2. Make sure the Node.js and Dart plugins are activated. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Dart Test Specific Configuration Settings

Item	Description

Test Mode	From this drop-down list, choose the scope of tests to execute. The available options are:
	– All in File: choose this option to to have IntelliJ IDEA run all the test methods in a file and specify this file in the Test File text box.
	– All in Folder: choose this option to to have IntelliJ IDEA run all the test methods from the files stored in a folder and specify this folder in the Test Folder text box.
	– Group or test by name: choose this option to to have IntelliJ IDEA run a specific tast or a group of tests.
	  – In the Test File text box, specify the file where the test or the group is defined.
	  – In the Group or Test Name text box, type the name of the test or the group of tests to execute.
	
	See more about grouping tests at https://www.dartlang.org/articles/dart-unit-tests/#grouping-tests .

Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button ... to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button + and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button –. – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the

CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Docker Deployment

Run | Edit Configurations | + | Docker Deployment

Docker Deployment run/debug configurations let you download and build Docker images, create and start Docker containers, and deploy your application artifacts into the containers.

To make Docker Deployment run/debug configurations available, you must download and install the Docker integration plugin , see Installing, Updating and Uninstalling Repository Plugins .

– Name, Share and Single instance only
– Deployment tab
– Container tab
– Before Launch options
– Toolbar

See also, Docker .

Name, Share and Single instance only

Item	Description
Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.

Deployment tab

Item	Description
Server	Select the Docker configuration to be used. To create a new configuration, or to edit an existing one, click … (`Shift+Enter`). For more information, see Docker.
Deployment	One of the following: – Docker Image. Select this option if you want to deploy a Docker image available locally. – <docker-dir>/Dockerfile. Select the Dockerfile to be used. (The Dockerfiles available in your project are suggested.) – <docker-dir>/docker-compose.yml. Select the Docker Compose YAML file to be used. (The docker-compose.yml files available in your project are suggested.)
The following settings are not available if a `docker-compose.yml` file is selected in the Deployment field.	
Image ID	If Docker Image is selected in the Deployment field: the ID of the image to be deployed, a sequence of hexadecimal symbols. For the image of interest, you can, for example, use the Copy image ID command in the Application Servers tool window and then paste the ID into this field.
Image tag	If <docker-dir>/Dockerfile is selected in the Deployment field: if you want to use an image with a particular tag, specify that tag. (The image in this case is specified in the Dockerfile.) If nothing is specified, Docker will look for the image with the tag `latest` .
Container name	If you want to give the container that will be created a particular name, specify that name. Otherwise, Docker will itself decide what the name should be.
After launch	Select this check box to start a web browser after starting the run/debug configuration. Select the browser from the list. Click … (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
Debug Port	The port to be used for debugging. (Debugging is supported only for Java.) This may be any unused port on your computer. If there is a warning *Debug port forwarding not found*, click Fix. As a result, the necessary port mapping info is added to the JSON container settings file . Note the command line arguments suggested by IntelliJ IDEA. To make debugging possible, these should be used to start the JVM in the container.

Container tab

This tab contains the settings for your container.

NOTE: The settings on this tab are ignored if a `docker-compose.yml` file is selected in the Deployment field .

Item	Description
JSON file	Path to a JSON file with container settings in the Docker Remote API format. – … lets you select an existing file. – + lets you generate and save a sample container settings `.json` file. CLI lets you translate the `docker run` options into JSON container settings Remote API format. The result is saved in a JSON file which is associated with your container. (For `docker run [OPTIONS] IMAGE [COMMAND] [ARG...]` `OPTIONS` are specified in the CLI options field of the dialog that opens when you click CLI,

`COMMAND` is specified in the Command field,

`ARG...` part is specified in the Entrypoint or Command filed.)

The settings below the JSON file field take precedence when overlapping ones set in the JSON file.	
Entrypoint	The entry point for your container, see e.g. ENTRYPOINT (default command to execute at runtime) in Docker run reference.
Command	See CMD (default command or options) in Docker run reference.
Publish all ports	If the check box is selected, all the container's exposed ports are bound to a random host port. See e.g. EXPOSE (incoming ports) in Docker run reference.
Port bindings	Container port/protocol - Host IP address/port mappings. E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine. For more info, see e.g. Connect using network port mapping in Linking containers together.
Links	Container name - alias mappings. See e.g. Communication across links in Linking containers together.
Volume bindings	Volume bindings such as container path - host path. See e.g. Managing data in containers.
Environment variables	Environment variables as variable - value pairs. See e.g. ENV (environment variables) in Docker run reference.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts. – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant. – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven. – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Debug tool window opens when you start the run/debug configuration in the debug mode. Otherwise, the tool window isn't shown. However, when the configuration is running in the debug mode, you can open the Debug tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click . See also, Creating Folders and Grouping Run/Debug Configurations.

Run/Debug Configuration: Firefox Remote

In this dialog box, create configurations to use for debugging applications locally in Firefox, versions 36 and higher. Note that remote debugging in Firefox is currently not supported at all.

On this page:

- Firefox Remote-Specific Configuration Settings
- Toolbar
- Common options

Firefox Remote-Specific Configuration Settings

Item	Description
Host	In this text box, specify the host where the application is running. Currently it is just `localhost`.
Port	In this spin box, specify the port the debugger will listen to. It must be the port that you specified when enabling debugging in Firefox, see Debugging JavaScript. The default value is 6000.

Toolbar

	Item	Shortcut	Description
+		`Alt+Insert`	Click this button to add a new configuration to the list.
−		`Alt+Delete`	Click this button to remove the selected configuration from the list.
		`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults		Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓		`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder		Use this button to create a new folder. If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations		Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.

Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.

- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Flash App

Use this dialog to create or edit Flash App run/debug configurations which let you run and debug your ActionScript and Flex applications.

Note that this run/debug configuration type is not available for build configurations whose output is a RLM or Library.

Also note that the settings depend on the target platform specified in the associated build configuration (Web, Desktop (AIR) or Mobile (AIR Mobile)).

- Name, Share and Single instance only
- Build configuration and Override main class
- Settings specific to Web-targeted applications
- Settings specific to Desktop-targeted applications (AIR)
- Settings specific to Mobile device-targeted applications (AIR Mobile)
- Before Launch options
- Toolbar

Name, Share and Single instance only

Item	Description
Name	Use this field to edit the name of the run/debug configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.

Build configuration and Override main class

Item	Description
Build configuration	Select the build configuration to be used.
Override main class	Normally, the main application class is defined in the build configuration. If you want a different class to be used as the main application class, select this option and specify the class. Use … (Shift+Enter) to select the necessary class in the Select Main Class dialog. Optionally, change the application output file name suggested by IntelliJ IDEA in the Output file name field.

Settings specific to Web-targeted applications

Item	Description
What to Launch	Select one of the following options: – Build output. Use this option to run the generated SWF file using the associated HTML wrapper. – URL or local file. Use this option to open a specified URL in a Web browser, or to run a specified local `.swf` file (either directly or using the corresponding `.html` wrapper). Type the desired URL in the field. Generally, this is going to be something like `http(s)://<host>:<port>/<context-root>`. In the case of a local file, you can use … and select the necessary `.swf` or `.html` file in the dialog that opens.
Launch with	Specify the program to be used for running your application. You can use: – The system default application, that is, the program associated with the target file type (HTML or SWF) in the operating system. Usually, this is a Web browser. For SWF files, this may also be a stand-alone Flash player. – A Web browser. – A Flash player. To select the required program, click … to the right of the Launch with field, and then specify the program in the Launch With dialog: – To select the system default application, just click System default application. – To select the browser, click Browser and select the required browser from the list. Additionally, you can access the Web Browsers dialog to adjust Web browser settings. To open this dialog, use … next to the list. – To select the Flash player, click Flash Player and then click … next to the Flash Player field. Then, specify the location of the required Flash player in the dialog that opens . – New instance (available only on OS X). Select the check box if you want a new instance of Flash Player to be started. Otherwise, if Flash Player is already running, a new window for the running instance will open.
Use debugger from SDK	Specify the Flex SDK that contains the debugger that should be used. (If you are using a Flex SDK 3 for compilation, a Flex SDK 4 is recommended for debugging.) If the corresponding SDK is already defined in IntelliJ IDEA, select the SDK from the list. Otherwise, click … (Shift+Enter) and add a definition of the necessary SDK in the Configure SDK dialog that opens.
Place SWF file in a local-trusted sandbox	If the Build output option is selected: select this check box if you want to register your application SWF file as trusted. Trusted SWF files can interact with any other SWF files. They can load data from anywhere, remote or local. Technically, trusted SWF files are assigned to the local-trusted sandbox.

Settings specific to Desktop-targeted applications (AIR)

Item	Description

AIR Debug Launcher options	If necessary, specify the AIR Debug Launcher options. Use the same rules as for specifying the program parameters .
Program parameters	Specify the parameters to be passed to the application. – Use spaces to separate individual parameters. – If a parameter includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .

Settings specific to Mobile device-targeted applications (AIR Mobile)

ItemDescription

Run on	Specify whether you want to use an emulator or a real Android or iOS device to run or debug your application: – Emulator. Select this option to use a built-in emulator. Select the intended target device from the list. The figures shown to the right of the list are the screen characteristics of the selected target device and are just for your information. – Screen is the screen size available to your application. – Full is the full screen size of the device. – ppi (pixels per inch) is the screen pixel density. If the necessary device is not present in the list, you can select Other and specify the desired screen parameters in the corresponding fields. – Android device. Select this option to run or debug your application using an Android mobile device. **Note** – To be able to use an Android device for running or debugging your application, packaging your application for Android must be enabled in the associated build configuration (the Enabled option on the Android tab). – Running or debugging an application using an Android device, normally, assumes installing the application package on the target device. For this operation to succeed, the mobile device must be connected to your computer using USB. – iOS Simulator. (This option is available only on Mac computers.) Select this option to run or debug your application using an iOS device simulator. In the SDK field, specify the path to the Apple iOS SDK to be used. (The simulator is included in an Apple iOS SDK.) You can click … (`Shift+Enter`) and select the SDK installation folder in the dialog that opens . **Note** To be able to use an iOS simulator for running or debugging your application, packaging your application for iOS must be enabled in the associated build configuration (the Enabled option on the iOS tab). – iOS device. Select this option to run or debug your application using an iOS mobile device. If necessary, enable fast application packaging. (If fast packaging is enabled, the ActionScript bytecode is interpreted and not translated to machine code. As a result, packaging is performed faster but code execution is slower.) **Note** – To be able to use an iOS device for running or debugging your application, packaging your application for iOS must be enabled in the associated build configuration (the Enabled option on the iOS tab). – Running or debugging an application using an iOS device, normally, assumes installing the application package on the target device. For this operation to succeed, the mobile device must be connected to your computer using USB. – If you are using the AIR SDK version 3.4 or later, the application package will be installed on the target device automatically. For earlier SDK versions, you'll have to initiate the installation manually.
Debug on device over	For an Android or iOS device: specify how the device will communicate with your computer after the application has been installed and started: – Network. Select this option if the device is going to communicate with your computer over the network. **Note** Installing the application package on a device requires a USB connection. – USB. Select this option if the device will be connected to your computer using USB. If necessary, change the port suggested by IntelliJ IDEA.
ADL options (emulator)	For the emulator: if necessary, specify the AIR Debug Launcher (ADL) options. Use the same rules as for specifying the program parameters .
App descriptor (emulator)	For the emulator: specify the application descriptor to be used. The available options refer to the descriptor-related settings in the associated build configuration. – as set for Android means the corresponding settings on the Android tab. – as set for iOS refers to the settings on the iOS tab. Depending on the settings in the build configuration, the following cases are possible: – as set for Android: <Android support is not enabled>. Generating the descriptor for Android is disabled. Use the Enabled check box on the Android tab if you want to enable the corresponding option. – as set for Android: generated. An auto-generated descriptor will be used. If you want a template-based descriptor to be used instead, select the Custom template option and specify the template. – as set for Android: <file_name>.xml. The corresponding custom template will be used to generate the descriptor. The `<file_name>` in this case is the name of the template file that will be used. The corresponding cases are also possible for the as set for iOS option.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration. The specified tasks are performed in the order that they appear in the list.

ItemShortcutDescription

+	`Alt+Insert` Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to be run. This option is available only if you have already at least one run/debug configuration in the current project.

- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located.
  For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .

	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Toolbar

Item	Description
(Insert)	Use this icon or shortcut to create a new run/debug configuration.
(Delete)	Use this icon or shortcut to delete the selected run/debug configuration.
(Ctrl+D)	Use this icon or shortcut to create a copy of the selected run/debug configuration.
	Click this button to edit the default settings for run/debug configurations.
	Use these buttons to move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.

Run/Debug Configuration: Flash Remote Debug

Use this dialog to create or edit Flash Remote Debug run/debug configurations which let you debug applications (SWF files) that have already been compiled and, if necessary, packaged, and are ready to be run on a local or remote computer, or a mobile device.

- Main settings
- Before Launch options
- Toolbar

See also, Using Flash Remote Debug configurations .

Main settings

Item	Description
Name	Use this field to edit the name of the run/debug configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.
Build configuration	Select the build configuration to be used. Note that only the following build configuration properties are relevant in the context of this run/debug configuration type: – Flex SDK. The debugger included in the corresponding SDK is used. – Dependencies. In addition to the module sources, the source files are also searched for among the build configuration dependencies for which source code is available.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration. The specified tasks are performed in the order that they appear in the list.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to be run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Toolbar

Item	Description
+ (Insert)	Use this icon or shortcut to create a new run/debug configuration.
– (Delete)	Use this icon or shortcut to delete the selected run/debug configuration.
(Ctrl+D)	Use this icon or shortcut to create a copy of the selected run/debug configuration.
	Click this button to edit the default settings for run/debug configurations.
	Use these buttons to move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.

Run/Debug Configuration: FlexUnit

Use this dialog to create or edit FlexUnit run/debug configurations which let you run or debug your FlexUnit tests.

Note that this run/debug configuration type cannot be used for running or debugging the tests on mobile devices. To run or debug the tests intended for mobile devices, you should use a built-in device emulator.

– Main settings
– Before Launch options
– Toolbar

Main settings

Item	Description
Name	Use this field to edit the name of the run/debug configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.
Build configuration	Select the build configuration to be used. Note that the corresponding build configuration should have a core FlexUnit library among its dependencies, see Testing ActionScript and Flex Applications .
Test	Select the test scope and specify the associated settings: – All in Package. Select this option to run all tests in a certain package. Specify the qualified package name in the Package and Method fields respectively. Use … (Shift+Enter) to select the desired package in the Choose Package and the Choose Test Method dialogs. To run all tests in all the packages of a module, leave the Package field empty. – Class or suite. Select this option to run a test class or suite. Specify the fully qualified name of the test class in the Class and Method fields respectively. Use … (Shift+Enter) to select the desired class in the Choose Test Class and the Choose Test Method dialogs. To run all tests in all the packages of a module, leave the Package field empty. – Method. Select this option to run a test method. Specify the fully qualified name of the test class and the method name in the Class and Method fields respectively. Use … (Shift+Enter) to select the desired class and method in the Choose Test Class and the Choose Test Method dialogs.
Show test log output	If you want the test log to be output, select this check box and select the logging level from the list. Note that a standard logger from the mx.logging package will be used.
Launch with	For Web build configurations: specify the program to be used for running your test. You can use: – The system default application, that is, the program associated with the target file type (HTML or SWF) in the operating system. Usually, this is a Web browser. For SWF files, this may also be a stand-alone Flash player. – A Web browser. – A Flash player. To select the required program, click … to the right of the Launch with field, and then specify the program in the Launch With dialog: – To select the system default application, just click System default application. – To select the browser, click Browser and select the required browser from the list. Additionally, you can access the Web Browsers dialog to adjust Web browser settings. To open this dialog, use … next to the list. – To select the Flash player, click Flash Player and then click … next to the Flash Player field. Then, specify the location of the required Flash player in the dialog that opens .
Place SWF file in a local-trusted sandbox	For Web build configurations: select this check box if you want to register your test SWF file as trusted. Trusted SWF files can interact with any other SWF files. They can load data from anywhere, remote or local. Technically, trusted SWF files are assigned to the local-trusted sandbox .

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration. The specified tasks are performed in the order that they appear in the list.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to be run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that

opens, specify where your CoffeeScript source files are located.
For more information, see CoffeeScript Support .

– Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
For more information, see Maven .

	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Toolbar

Item	Description
(Insert)	Use this icon or shortcut to create a new run/debug configuration.
(Delete)	Use this icon or shortcut to delete the selected run/debug configuration.
(Ctrl+D)	Use this icon or shortcut to create a copy of the selected run/debug configuration.
	Click this button to edit the default settings for run/debug configurations.
	Use these buttons to move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.

Run/Debug Configuration: Geronimo Server

Run | Edit Configurations | + | Geronimo Server | Local or Remote

Geronimo Server run/debug configurations let you deploy and debug your applications on Apache Geronimo . (The Geronimo Integration plugin must be enabled.)

See the server documentation to find out which JRE version is compatible with the version of Geronimo that you are using. Also note that Geronimo needs the `JAVA_HOME` or the `JRE_HOME` environment variable to be set.

- Name field and Share option
- Server tab for a local configuration
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a local configuration
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.

On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Username	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the server.
Password	The password of the user specified in the Username field .

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
JMX Port	The server JMX port.
Username	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the server.
Password	The password of the user specified in the Username field .
Host	The fully qualified domain name or the IP address of the server host.
Port	The server HTTP port.

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. For more information, see the table below.

Item	Description
+(Alt+Insert)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
–(Alt+Delete)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(F4)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point

	to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
−	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use + or + to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click −.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a local configuration

Item	Description
Run / Debug / Coverage	Use to switch between the settings for the run, debug and code coverage modes.
Startup script	Specify the script to be used to start the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Startup Script dialog which shows the contents of the Startup script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click … and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Specify the script to be used to stop the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Shutdown Script dialog which shows the contents of the Shutdown script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click … and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Pass environment variables	To pass specific variables to the server environment, select this check box and specify the variables: – To add a variable, click + and specify the variable name and value in the Name and Value fields respectively. – To remove a variable from the list, select the variable and click −.
Port	Use this field to change the debugger port.
Debugger Settings	Click this button to edit the debugger options on the Debugger page of the Settings dialog .

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click . See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: GlassFish Server

Run | Edit Configurations | + | GlassFish Server | Local or Remote

GlassFish Server run/debug configurations let you deploy and debug your applications on GlassFish Server . (The GlassFish Integration plugin must be enabled.)

- Name field and Share option
- Server tab for a local configuration
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a local configuration
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .

JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Server Domain	Select the server domain to be used (to deploy and run your application).
Username	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the server.
Password	The password of the user specified in the Username field .
Preserve Sessions Across Redeployment	For GlassFish Server 3 or later versions: select this check box to preserve active HTTP sessions when redeploying your application artifacts.

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Domain Admin Server Port	GlassFish domain administration server port.
Username	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the server.
Password	The password of the user specified in the Username field .
Deploy Target	Specify the following information for the target server: – Domain. Select this option if the server operates in the domain mode. Virtual server. To deploy onto a particular virtual server in the target domain, specify the name of the virtual server. Otherwise, the artifacts are deployed onto all the virtual servers in the domain. – Cluster. Select this option if the server operates in the cluster mode. Cluster Name. Specify the name of the target cluster. Domain Admin Server Host. Specify the fully qualified domain name or the IP address of the domain administration server host.
Use SSL connection	Select this check box to connect to the server using SSL.
Preserve Sessions Across Redeployment	For GlassFish Server 3 or later versions: select this check box to preserve active HTTP sessions when redeploying your application artifacts.
Upload with GlassFish	Select this check box to deploy the application artifacts using the GlassFish Admin Console. Note that in this case you won't be able to deploy exploded artifacts to a remote server (i.e. if the server is actually on a different computer). If you use this option, you don't need to specify the remote staging settings.
Remote staging	This section contains the settings related to staging . An example of remote staging settings for a mounted folder is provided after this table.
Type	Select the way the staging environment or host is accessed for transferring the application artifact or artifacts from your local computer. (In the user interface of IntelliJ IDEA this setting is also referred to as the connection type.) The available options are: – Same file system. Select this option if the target server is installed on your local computer. The artifacts in this case are deployed locally and, thus, don't need to be transferred to a remote host. – ftp. The File Transfer Protocol or Secure FTP is used. – Local or mounted folder. The staging environment is a local folder or is accessed as a mounted folder . If the list is empty, you have to enable the Remote Hosts Access plugin which supports the corresponding functionality.
Host	If Same file system is selected for Type, the only available option for Host is also Same file system. In all other cases, the list contains the existing configurations of the selected type . So each configuration corresponds to an individual (S)FTP connection, or a local or mounted folder. Select an existing configuration or create a new one. To create a new configuration: 1. Click ... to the right of the list. 2. In the Deployment dialog , click +. 3. In the Add Server dialog, specify the configuration name, select the type, and click OK.

4. On the Connection tab , specify the settings in the Upload/download project files section.
   The rest of the settings don't matter.
5. Click OK in the Deployment dialog.

Staging	When deploying to the remote host, the application artifact or artifacts are placed into a staging folder which should be accessible to GlassFish Server. The settings in this section define the location of this staging folder. Note that if Same file system is selected for Type and Host, no settings in this section need to be specified.
Path from root	The path to the staging folder relative to the local or mounted folder, or the root of the (S)FTP host. You can use ... to select the folder in the Choose target path dialog.
Mapped as	The absolute path to the staging folder in the local file system of the remote host.
Remote connection settings	The settings for accessing deployed applications.
Host	The fully qualified domain name or the IP address of the GlassFish Server host. When the target server operates in the cluster mode, this is the host of the GlassFish Server instance which the debugger should connect to.
Port	The server HTTP port, or, for a cluster, the HTTP port of the corresponding GlassFish Server instance.

An example of remote staging settings for a mounted folder

Assuming that:

- `C:\shared` is a shared folder on the remote host which is mounted to the local computer as the drive `X:` .
- The folder that you are going to use for staging is `C:\shared\staging` .

Here are the corresponding remote staging settings:

- Type: Local or mounted folder.
- Host: The configuration should be selected in which the value in the Folder field is `X:\` (the Upload/download project files section on the Connection tab of the Deployment dialog).
- Staging/Path from root: `staging`
- Staging/Mapped as: `C:\shared\staging`

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. To edit the settings for an artifact or external resource, select the corresponding item in the list and use the controls in the right-hand part of the tab. For more information, see the table below.

Item	Description
+(`Alt+Insert`)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
–(`Alt+Delete`)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(`F4`)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)
Use custom context root	If you want to assign a particular context root to an artifact or external resource, select the artifact or the resource, select the check box and specify the desired context root in the field underneath the check box.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .

—	Click this button to remove the selected log entry from the list.
…	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use + or + to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click —.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a local configuration

Item	Description
Run / Debug / Coverage	Use to switch between the settings for the run, debug and code coverage modes.
Startup script	Specify the script to be used to start the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Startup Script dialog which shows the contents of the Startup script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click … and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Specify the script to be used to stop the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Shutdown Script dialog which shows the contents of the Shutdown script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click … and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .

Pass environment variables	The way GlassFish Server is started cannot be changed by using environment variables. The server configuration file `domain.xml` should be changed instead. So, whatever you specify in this section, doesn't matter.
	To make the necessary changes to the configuration file, IntelliJ IDEA provides quick fixes. When you see a warning in the lower part of the dialog that certain settings are incorrect, you should click Fix. As a result, the necessary changes are made to `domain.xml` .
	
	Example
	
	When you select Debug in the upper part of the tab, you see that there is a variable `JAVA_OPTS` with the value
	
	`-agentlib:jdwp=transport=dt_socket,address=127.0.0.1:9009,suspend=y,server=n`
	
	As stated before, the corresponding setting cannot be passed to the GlassFish Server JVM by means of `JAVA_OPTS` .
	
	In the configuration file `<domain_name>/config/domain.xml` , by default, the corresponding value (the `debug-options` attribute of the `<java-config>` element) is
	
	`-agentlib:jdwp=transport=dt_socket,address=9009,server=y,suspend=n`
	
	So, in the lower part of the dialog, you see the warning *Debug settings are invalid*....
	
	Now, if you click Fix, the value in `domain.xml` changes to
	
	`-agentlib:jdwp=transport=dt_socket,address=9009,server=n,suspend=y`

Port	Use this field to change the debugger port.

	When the warning *Debug settings are invalid...* is shown, click Fix. This changes the port number in the GlassFish Server configuration file `domain.xml`. See also, Pass environment variables.
Debugger Settings	Click this button to edit the debugger options on the Debugger page of the Settings dialog.

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These should be specified in the GlassFish Server configuration file `domain.xml`.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts. – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant. – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven. – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click . See also, Creating Folders and Grouping Run/Debug Configurations.

Run/Debug Configuration: Google App Engine Deployment

Run | Edit Configurations | + | Google App Engine Deployment

Google App Engine Deployment run/debug configurations let you deploy your application artifacts to Google App Engine . (The Google App Engine Integration plugin must be enabled.)

– Main settings
– Before Launch options
– Toolbar

Main settings

Item	Description
Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.
Server	Select the cloud access configuration to be used. To create a new configuration, or to edit an existing one, click ... (Shift+Enter). For more information, see Google App Engine.
Deployment	Select the application artifact to be deployed. Note that only the WAR exploded artifact format can be deployed.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Debug tool window opens when you start the run/debug configuration in the debug mode. Otherwise, the tool window isn't shown. However, when the configuration is running in the debug mode, you can open the Debug tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder.

To remove grouping, select a folder and click —.

See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: Gradle

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Gradle-specific options

Item	Description
Gradle Project	Use this field to specify the location of your Gradle project. You can either enter it manually or click the Browse button and point to the desired location in the dialog that opens . You can also click button to select an available Gradle module from the list of registered Gradle modules in your existing IntelliJ IDEA project. The list has a tree structure that might be useful if you have a Gradle multi-module project.
Tasks	Use this field to specify external tasks for your Gradle project. Use spaces to separate one task from another.
VM Options	Use this field to specify VM options for your Gradle project. If you need more room to type, click next to the field to access the VM options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts .

- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Grails

Grails run/debug configuration enables you to run and debug the Grails applications, tests and Web tests.

The dialog box consists of the following tabs:

- Grails tab
- Code Coverage tab
- Maven Settings tab

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Grails Tab

Item	Description
Module	Select application, for which this run/debug configuration is created. By default, the name of the current module is suggested.
Command line	Type a command to execute a particular target, for example, `run-app` , or `app-engine` . Alternatively, you can execute target as described in the section Running Grails Targets .
VM Options	Specify the string to be passed to the VM for launching the application. This string may contain the options such as `-mx` , `-verbose` , etc. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Environment Variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values.
Add --classpath	If this check box is selected, it means that the user intends to include the dependency directly, by passing `--classpath` to the command line. Please note the following: – This check box is enabled, if a module has a dependency that was added directly from the Dependencies tab of the Project Structure dialog, without changing `BuildConfig.groovy` or plugins dependencies. – Read about adding module dependencies in the section Configuring Module Dependencies and Libraries .
Launch browser	Select this check box to open the default browser.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Maven Settings Tab

Use this tab to configure Maven settings for running and debugging your application. By default, the Use project settings check box is selected and IntelliJ IDEA uses the default settings specified in your project.

Item	Description
Work offline	If this option is checked, Maven works in offline mode and uses only those resources that are available locally. This option corresponds to the `--offline` command line option.
Use plugin registry	Check this option to enable referring to the Maven Plugin Registry. This option corresponds to the `--no-plugin-registry` command line option.
Execute goals recursively	If this option is cleared, the build does not recur into the nested projects. Clearing this option equals to `--non-recursive` command line option.

Print exception stack traces	If this option is checked, exception stack traces are generated. This option corresponds to the `--errors` command line option.
Always update snapshots	Select this check box to always update snapshot dependencies.
Output level	Select the desired level of the output log, which allows plugins to create messages at levels of *debug*, *info*, *warn*, and *error*, or disable output log.
Checksum policy	Select the desired level of checksum matching while downloading artifacts. You can opt to fails downloading, when checksums do not match (`--strict-checksums`), or issue a warning (`--lax-checksums`).
Multiproject build fail policy	Specify how to treat a failure in a multiproject build. You can choose to: – Fail the build at the very first failure, which corresponds to the command line option `--fail-fast` . – Fail the build at the end, which corresponds to the command line option `--fail-at-end` . – Ignore failures, which corresponds to the command line option `--fail-never` .
Plugin update policy	Select plugin update policy from the drop-down list. You can opt to: – Check for updates, which corresponds to the command line option `--check-plugin-updates` . – Suppress checking for updates, which corresponds to the command line option `--no-plugin-updates` .
Threads `(-T option)`	Use this field to set the `-T` option for parallel builds. This option is available for Maven 3 and later versions. For more information, see parallel builds in Maven 3 feature.
Maven home directory	Use this drop-down list to select a bundled Maven version that is available (for Maven2, version 2.2.1 and for Maven3, version 3.0.5) or the result of resolved system variables such as `MAVEN_HOME` or `MAVEN2_HOME` . You can also specify your own Maven version that is installed on your machine. You can click … and select the necessary directory in the dialog that opens .
User settings file	Specify the file that contains user-specific configuration for Maven in the text field. If you need to specify another file, check the Override option, click the ellipsis button and select the desired file in the Select Maven Settings File dialog.
Local repository	By default, the field shows the path to the local directory under the user home that stores the downloads and contains the temporary build artifacts that you have not yet released. If you need to specify another directory, check the Override option, click the ellipsis button and select the desired path in the Select Maven Local Repository dialog.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select

the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools .

- Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise.
  If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration.
- Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result.
- Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration. Griffon

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Item	Description
Module	Select application, for which this run/debug configuration is created. By default, the name of the current module is suggested.
Command line	Type a command to execute a particular target, for example, `run-app` , or `app-engine` . Alternatively, you can execute target as described in the section Running Grails Targets .
VM Options	Specify the string to be passed to the VM for launching the application. This string may contain the options such as `-mx` , `-verbose` , etc. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Environment Variables	Click the Browse button ... to open the Environment Variables dialog box, where you can create variables and specify their values.
Add --classpath	If this check box is selected, it means that the user intends to include the dependency directly, by passing `--classpath` to the command line. Please note the following: – This check box is enabled, if a module has a dependency that was added directly from the Dependencies tab of the Project Structure dialog, without changing `BuildConfig.groovy` or plugins dependencies. – Read about adding module dependencies in the section Configuring Module Dependencies and Libraries .
Launch browser	Select this check box to open the default browser.

Toolbar

	Item	Shortcut	Description
+		`Alt+Insert`	Click this button to add a new configuration to the list.
–		`Alt+Delete`	Click this button to remove the selected configuration from the list.
		`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults		Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓		`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder		Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations		Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools .

- Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise.
  If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration.
- Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result.
- Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Groovy

This run/debug configuration is intended for the Groovy scripts .

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Item	Description
Script path	Type the path to the Groovy script to be launched, or click the ellipsis button and select one from the tree view.
Module	Select the module the run/debug configuration is created for. By default, the name of current module is suggested.
VM Options	Specify the string to be passed to the VM for launching the Groovy script. This string may contain the options such as `-mx` , `-verbose` , etc. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Script Parameters	Type the list of arguments to be passed to the Groovy Script, same way as if you were entering these parameters in the command line. Use the same rules as for specifying the VM options .
Environment variables	Click the Browse button … to open the Environment Variables dialog box, where you can create variables and specify their values.
Enable debug stacktrace	If this check box is selected, the detailed debug stacktrace is produced, including the Groovy method calls. This information can be helpful in case of exceptions.
Working directory	Specify the root directory for the files referenced in script.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration.

- Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result.
- Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Grunt.js

In this dialog box, create configurations for running Grunt.js tasks.

On this page:

- Getting Access to the Run/Debug Configuration: Grunt Dialog Box
- Grunt-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: Grunt Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS runtime environment that contains the Node Package Manager(npm) .
3. Install Grunt as described in Installing Grunt .

Grunt-Specific Configuration Settings

Item	Description
Gruntfile	In this field, specify the location of the `Gruntfile.js` file to retrieve the definitions of the tasks from. Select the path from the drop-down list or click the ... button and choose the file from the dialog box that opens.
Tasks	In this field, specify the tasks to run. Do one of the following: – To run one task, select it from the drop-down list. – To run several tasks, type their names in the text box using blank spaces as separators.
Force execution	Select this check box to have Grunt ignore warnings and continue executing the launched task until the task is completed successfully or an error occurs, If the check box is cleared, the task execution is stopped by the first reported warning.
Verbose mode	Select this check box to have the verbose mode applied and thus have a full detailed log of a task execution displayed.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node Options	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. See Node Parameters for details.
Grunt-cli Package	In this field, specify the path to the globally installed `Grunt-cli` package installed globally See Installing Grunt.js for details.
Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button ... to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button + and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button –. – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
–	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.

Share

Select this check box to make the run/debug configuration available to other team members.
If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise.

If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Gulp.js

In this dialog box, create configurations for running Gulp.js tasks.

On this page:

- Getting Access to the Run/Debug Configuration: Gulp.js Dialog Box
- Gulp.js-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: Gulp.js Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS runtime environment that contains the Node Package Manager(npm) .
3. Install the `Gulp` package as described in Installing Gulp.js .

Gulp.js-Specific Configuration Settings

Item	Description
Gulpfile	In this field, specify the location of the `Gulpfile.js` file to retrieve the definitions of the tasks from. Select the path from the drop-down list or click the ... button and choose the file from the dialog box that opens.
Tasks	In this field, specify the tasks to run. Do one of the following: – To run one task, select it from the drop-down list. – To run several tasks, type their names in the text box using blank spaces as separators.
Arguments	In this text box, specify the arguments for tasks to be executed with. Use the following format: `--<parameter_name> <parameter_value>` For example: `--env development` . For details about passing task arguments, see https://github.com/gulpjs/gulp/blob/master/docs/recipes/pass-arguments-from-cli.md
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node Options	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. See Node Parameters for details. In the default configuration, type `--harmony` in this text box to have IntelliJ IDEA build a tasks tree according to a `Gulfile.js` written in ECMA6. Technically, IntelliJ IDEA invokes Gulp.js and processes `Gulpfile.js` according to the default Gulp.js run configuration. However this is done silently and does not require any steps from your side. However, if your `Gulpfile.js` is written in ECMA6 , by default IntelliJ IDEA does not recognize this format and fails to build a tasks tree. To solve this problem, specify `--harmony` as a Node parameter of the default Gulp.js run configuration.
Gulp Package	In this field, specify the path to the `Gulp` package installed locally, under the project root. See Installing Gulp.js for details.
Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button ... to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button + and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button —. – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.

Toolbar

	Item	Shortcut	Description
+		`Alt+Insert`	Click this button to add a new configuration to the list.
—		`Alt+Delete`	Click this button to remove the selected configuration from the list.
		`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults		Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓		`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder		Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.

	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: GWT

Run | Edit Configurations

Shift+Alt+F10

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Item	Description
Module	From this drop-down list, select the IntelliJ IDEA module to apply this configuration to.
Use Super Dev Mode	Select this check box to use GWT Super Dev Mode . In this case you can quickly recompile your code and check the output in a browser.
GWT Modules to Load	From this drop-down list, select the GWT module to deploy.
VM options	In this text box, specify the string to be passed to the VM. Usually this string contains the options such as `-mx` , `-verbose` , etc. Type the arguments right in the text box or in the VM Options dialog that opens when you click . When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Dev Mode parameters	In this text box, specify the parameters for the GWT development mode . Type the parameters right in the text box or in the GWT Dev Mode Parameters dialog that opens when you click . Use the same rules as for specifying the VM options .
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens . Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. Note The list of environment variables may vary depending on the enabled plugins.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values.
Server	Use this drop-down list to select a server for your application. IntelliJ IDEA uses Jetty Web server as a default one.
Start Page	From this drop-down list, select the name of the `.html` file that implements the starting page of the application. This drop-down is still available even if you clear the Open in browser check box to run the application in the embedded GWT development mode without launching the browser.
JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Open in browser	Select this check box to have IntelliJ IDEA launch the browser and show the application implemented in the specified GWT module. From the drop-down list, choose the browser to open. The preselected Default value means that the browser specified as default at the IntelliJ IDEA will be launched. Contrary to previous IntelliJ IDEA versions, where this functionality was implemented through integration with the native GWT Development Mode window , now you can choose yourself whether to have the application opened in the browser or not.
with JavaScript debugger	Select this check box for IntelliJ IDEA to use a JavaScript debugger for your application. Note that IntelliJ IDEA will automatically use JetBrains Chrome Extension to debug your application in a browser.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.

Share

Select this check box to make the run/debug configuration available to other team members.
If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise.

If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Heroku Deployment

Run | Edit Configurations | + | Heroku Deployment

Heroku Deployment run/debug configurations let you deploy your code and application artifacts to Heroku . They also let you debug your applications.

For Heroku Deployment run/debug configurations to be available, the Heroku integration plugin must be enabled.

– Main settings
– Before Launch options
– Toolbar

Main settings

Item	Description
Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.
Server	Select the cloud access configuration to be used. To create a new configuration, or to edit an existing one, click ... (Shift+Enter). For more information, see Heroku.
Deployment	To deploy your source code, select the corresponding module . To deploy an application artifact , select the artifact. Only archive artifact formats can be used (e.g. WAR, EAR).
Use custom application name	By default, your application will have about the same name as the module or the artifact. To specify a different name, select the check box and specify the name in the field. (The application name defines its URL, `https://<app-name>.herokuapp.com/` .)
Debug Host	An Internet routable IP address of your computer or that of a router that redirects the debug connection from Heroku onto your local host.
Debug Port	The port to be used for debugging. This may be any unused port on your computer.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Debug tool window opens when you start the run/debug configuration in the debug mode. Otherwise, the tool window isn't shown. However, when the configuration is running in the debug mode, you can open the Debug tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.

	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click . See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: J2ME

J2ME run/debug configuration options vary depending on the SDK you are using to develop your mobile applications. This section describes the options that are specific for WTK and DoJa, as well as the general options that apply to both SDKs.

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Item	Description
Name	In this text box, specify the name of the current run/debug configuration.

WTK-specific options

Item	Description
Jad/Class/OTA	Select, which of the following sources - Jad, MIDlet class or OTA - is used to be run/debugged. Each source has its own settings described below.
JAD	If you have selected to use Jad file as the source for running and debugging, you need only to specify its location in the Jad File.
Class Settings	If you have selected to use MIDlet class as the source for running and debugging, specify the following options: – MIDlet Class: Specify here the MIDlet class to run/debug. – User Defined Settings: Use this options group to view/introduce/delete/move MIDlet class settings defined by the user.
OTA Settings	These options can be set, if you have selected to use OTA as the source for running and debugging. This functionality emulates wireless download of the software from the server. Therefore, it is necessary to have a working server with such service and the necessary suite (which you want to install and launch) on it. This service is completely dependent on the emulator vendor and IntelliJ IDEA does not provide it. So, it should be provided by the emulator or separately by the emulator vendor (for instance, like Nokia does). Thus, you need to have three items ready - a server, a service and tested suite on that server. Remember that the actual options list varies from emulator type. – Install from URL: Specify the URL from which the necessary suite can be downloaded. Later on, this suite is installed on the emulator. – Force installation: Select this option to reinstall the suite, if it is already installed. – Install, run and remove: The suite is installed on the emulator, launched and removed afterwards. – Run/Remove previously installed: The previously installed suites are run or removed, respectively. – Update Suites: Click this button to update the list of suites present on the server.

DoJa-Specific Options

Item	Description
Jam/Class	Select which of the files, Jam or IApplication Class, is to be run/debugged.
JAM	If you have selected to use Jam file as the source for running and debugging, you need only to specify its location in the Jam File field.
Class Settings	If you want to use IApplication Class as source for running/debugging, specify its location in the IApplication Class field.

General Options

Item	Description
Device	Select one of the available devices for the current emulator.
Open Preferences	Opens the emulator's Preferences dialog. For more information, refer to emulator's documentation.
Open Utils	Opens the emulator's Utilities dialog. For more information, refer to emulator's documentation.
Emulator parameters	Type in/edit emulator parameters such as `-mx heap_size` or `traceall` .
Use classpath and JDK of module	Select a mobile module from the list of modules configured in your project. The classpath and JDK of this module will be used to run your J2ME module with the current run configuration.

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
–	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in

the right-hand pane. The defaults are applied to all newly created run/debug configurations.

Share

Select this check box to make the run/debug configuration available to other team members.
If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise.

If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Jar Application

This run/debug configuration enables you to run applications started via `java -jar <name>.jar` command.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

Click here for the description of the options that are common for all run/debug configurations.

Configuration tab

Item	Description
Path to JAR	Specify here the filly-qualified path to the required JAR file.
VM options	In this text box, specify the string to be passed to the VM for launching an application. Usually this string contains the options such as `-mx`, `-verbose`, etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options.
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens. Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. Note The list of environment variables may vary depending on the enabled plugins.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values. Note that you can copy-paste the contents of the Environment variables field without having to open the Environment Variables dialog box.
JRE	By default, the newest JDK from the module dependencies is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Search sources using module's class path	Use the drop-down list to choose the required module. This option tells the debugger and the feature Navigate from stacktrace, where the source code for the classes from JAR archive should be sought for.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results.
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.

	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.

	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Java Scratch

The Java Scratch run/debug configuration enables you running or debugging Java scratch files that have the `main()` method.

Click here for the description of the options that are common for all run/debug configurations.

Item	Description
Main class	In this text box, specify the fully qualified name of the class to be executed (passed to the JRE). Type the class name manually or click the Browse button to open the Choose Main Class dialog box, where you can find the desired class by name or search through the project.
Path to scratch file	Specify here the path to the scratch file, or select one from the file chooser.
VM options	In this text box, specify the string to be passed to the VM for launching an application. Usually this string contains the options such as `-mx`, `-verbose`, etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options.
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens. Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. **Note** The list of environment variables may vary depending on the enabled plugins.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values. Note that you can copy-paste the contents of the Environment variables field without having to open the Environment Variables dialog box.
Use classpath of module	Select the module whose classpath should be used to run the application.
JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Enable capturing form snapshots	Select this check box to enable the GUI Designer to take snapshots of the GUI components, that can be afterwards converted into a form.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder. If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: JavaScript Debug

In this dialog box, create a configuration to be used for debugging JavaScript sources locally and remotely and for debugging Dart Web applications.

1. Local debugging. In this mode, the application files remain where they are in the IntelliJ IDEA project on you machine. IntelliJ IDEA runs them on the built-in Web server with the root URL `http://localhost:<built-in server port>/<project root>` . The built-in server port (1024 or higher) is specified on the JavaScript page of the Settings dialog box. A debugging session in this mode can be initiated in two ways:
   - Open the HTML file with the JavaScript injection to be tested and launch the temporary run/debug configuration that IntelliJ IDEA has generated automatically. This approach it helpful when you do not need to debug the entire application but just one script.
   - Create and launch a permanent debug configuration.
2. Remote debugging. In this mode, the application files are on an external web server and you have their copies in a IntelliJ IDEA project on your computer. No matter, whether the web server itself is running on a physically remote host or on your machine, application files deployed on it are treated as remote. For details, see Deployment. Working with Web Servers .
   You can deploy the application files to a web server and launch the application yourself or debug an already running application.

   When a remote HTML file with a JavaScript injection is opened, the debugger tells IntelliJ IDEA the name of the currently processed file and the number of the line to be processed. IntelliJ IDEA opens the local copy of this file and indicates the line with the provided number. This behaviour is enabled by specifying correspondence between files and folders on the server and their local copies. This correspondence is called mapping, it is set in the debug configuration.

   A Remote debugging session can be initiated only through a permanent debug configuration that you have to define yourself manually.

IntelliJ IDEA supports local and remote JavaScript debugging only in Google Chrome and other browsers of the Chrome family. In Firefox , remote debugging is not supported at all. Local debugging is supported for Firefox, version 36 an higher, through the Firefox Remote debug configuration.

IntelliJ IDEA supports local and remote JavaScript debugging only in Google Chrome and other browsers of the Chrome family. In Firefox , remote debugging is not supported at all. Local debugging is supported for Firefox, version 36 an higher, through the Firefox Remote debug configuration.

On this page:

- JavaScript Debug-Specific Configuration Settings
- Toolbar
- Common options

JavaScript Debug-Specific Configuration Settings

Item	Description
Name	In this text box, specify the name of the current run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
URL	– Debugging JavaScript: In this text box, specify the URL address of the HTML file that references the JavaScript to debug. For local debugging, type the URL in the format `http://localhost:<built-in server port>/<project root>` . The built-in server port (1024 or higher) is specified on the JavaScript page of the Settings dialog box. – Debugging a Dart Web application: In this text box, specify the URL address of the HTML file the references the Dart code to debug. Make sure that the port specified in this URL address is the same as the Built-in server port on the Debugger page of the Settings dialog box and the port specified in the settings for the Chrome Extension.
Browser	From this drop down list, select the browser, where your application will be debugged. – Debugging JavaScript: Chrome – Debugging a Dart Web application: If you choose Dartium which has a built-in Dart virtual machine, simply the Dart code is executed. If you choose Chrome, the Dart code is compiled into JavaScript through the Dart2js tool, which is a part of the Pub tool. Whether such compilation is required or not is decided by the Pub Serve tool. The tool is invoked automatically when you run or debug a Dart web application, either by opening an HTML file or by launching a run/debug configuration. The tool analyzes the `<script></script>` element in the HTML file. If the element references a Dart file, Pub Serve invokes the Dart2js compiler whereupon the compiled JavaScript files are passed to the browser where they are processed. Specify the URL address of the HTML file the references the Dart code to debug. Make sure that the port specified in this URL address is the same as the Built-in server port on the Debugger page of the Settings dialog box and the port specified in the settings for the Chrome Extension.
Remote URLs of local files	– Debugging JavaScript: IntelliJ IDEA displays this area only when you create a permanent debug configuration manually. For automatically generated temporary configurations the area is not shown. In this area, map the local files to be involved in debugging to the URL addresses of their copies on the server. – File/Directory - in this read-only field, select the desired local file or directory in the project tree. – Remote URL - in this text box, type the absolute URL address of the corresponding file or folder on the server. These mappings are required only if the local folder structure under the project root differs from the folder structure on the server. If the structures are identical, IntelliJ IDEA itself "retrieves" the paths to local files by parsing the URL addresses of their copies on the server. – Debugging a Dart Web application: IntelliJ IDEA displays this area only when the port specified in the URL field is different from the port of the built-in Web server specified on the Debugger page of the Settings dialog box.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.

	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.

	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: JBoss Server

Run | Edit Configurations | + | JBoss Server | Local or Remote

JBoss Server run/debug configurations let you deploy and debug your applications on JBoss Server . (The JBoss Integration plugin must be enabled.)

- Name field and Share option
- Server tab for a local configuration
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a local configuration
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.

VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start.
	If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger.
	
	When specifying the options, follow these rules:
	– Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` .
	– If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` .
	– If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
	
	Here is how you can specify a custom configuration directory and file name:
	`-Djboss.server.config.dir=C:/jboss-eap-6.3/custom-configuration`
	`-Djboss.server.default.config=standalone-custom.xml`

On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function.

	The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Username	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the server.
Password	The password of the user specified in the Username field .
Operating mode	Select: – standalone to start JBoss as a standalone server. Port offset. If necessary, specify the port offset, i.e. the number by which all the server ports should be shifted. (The ports are specified by means of the `<socket-binding>` elements in the server configuration file `standalone.xml` .) The specified offset is passed to the server as a VM option, e.g. `-Djboss.socket.binding.port-offset=100` – domain to start the server as a managed domain. Server group. Select the server group to deploy your artifacts to. (The list items correspond to the `<server-group>` elements in the configuration file `domain.xml` . The server group doesn't matter if you don't deploy anything using this run configuration.) When you select other-server-group, the following message may be shown in the lower part of the dialog: *Error: No single server is configured to start for server group 'other-server-group'* . If this is the case and you click Fix, the configuration file `host.xml` is modified. (The `auto-start` attribute of the first `<server>` element in the server group is changed from `false` to `true` .)

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Management port	The native management interface port. (This port is defined in the `standalone.xml` configuration for a standalone server and in the `host.xml` configuration for a server within a managed domain.)
Operating mode	The server operating mode: standalone for a standalone server and domain for a managed domain. For the managed domain mode, specify the server group to deploy your artifacts to, e.g. `main-server-group` . The server group doesn't matter, if you don't deploy anything using this run configuration.
Username	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the server.
Password	The password of the user specified in the Username field .
Remote staging	This section contains the settings related to staging . An example of remote staging settings for a mounted folder is provided after this table.
Type	Select the way the staging environment or host is accessed for transferring the application artifact or artifacts from your local computer. (In the user interface of IntelliJ IDEA this setting is also referred to as the connection type.) The available options are: – Same file system. Select this option if the target server is installed on your local computer. The artifacts in this case are deployed locally and, thus, don't need to be transferred to a remote host. – ftp. The File Transfer Protocol or Secure FTP is used. – Local or mounted folder. The staging environment is a local folder or is accessed as a mounted folder . If the list is empty, you have to enable the Remote Hosts Access plugin which supports the corresponding functionality.
Host	If Same file system is selected for Type, the only available option for Host is also Same file system. In all other cases, the list contains the existing configurations of the selected type . So each configuration corresponds to an individual (S)FTP connection, or a local or mounted folder. Select an existing configuration or create a new one. To create a new configuration:

1. Click ... to the right of the list.
2. In the Deployment dialog , click +.
3. In the Add Server dialog, specify the configuration name, select the type, and click OK.
4. On the Connection tab , specify the settings in the Upload/download project files section.
   The rest of the settings don't matter.
5. Click OK in the Deployment dialog.

Staging	When deploying to the remote host, the application artifact or artifacts are placed into a staging folder which should be accessible to JBoss Server. The settings in this section define the location of this staging folder. Note that if Same file system is selected for Type and Host, no settings in this section need to be specified.
Path from root	The path to the staging folder relative to the local or mounted folder, or the root of the (S)FTP host. You can use ... to select the folder in the Choose target path dialog.
Mapped as	The absolute path to the staging folder in the local file system of the remote host.
Remote connection settings	The settings for accessing deployed applications.
Host	The fully qualified domain name or the IP address of the JBoss Server host. When the target server operates in the managed domain mode, this is the Domain Controller host.
Port	The server HTTP port.

An example of remote staging settings for a mounted folder

Assuming that:

- `C:\shared` is a shared folder on the remote host which is mounted to the local computer as the drive `X:`.
- The folder that you are going to use for staging is `C:\shared\staging`.

Here are the corresponding remote staging settings:

- Type: Local or mounted folder.
- Host: The configuration should be selected in which the value in the Folder field is `X:\` (the Upload/download project files section on the Connection tab of the Deployment dialog).
- Staging/Path from root: `staging`
- Staging/Mapped as: `C:\shared\staging`

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. For more information, see the table below.

Item	Description
+(`Alt+Insert`)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens . For a Web Application Exploded artifact, the following message may be shown in the lower part of the dialog: *Error: Artifact '<name>' has invalid extension.* If this is the case, click Fix and add `.war` at the end of the output directory name.
–(`Alt+Delete`)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
↑(`Alt+Up`)	Use this icon or shortcut to move the selected item one line up in the list.
↓(`Alt+Down`)	Use this icon or shortcut to move the selected item one line down in the list.
(`F4`)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .

Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
−	Click this button to remove the selected log entry from the list.
...	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use or to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click −.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a local configuration

Item	Description
Run / Debug / Coverage	Use to switch between the settings for the run, debug and code coverage modes.
Startup script	Specify the script to be used to start the server. Use default: − If this check box is selected, the default script is used. in this case opens the Default Startup Script dialog which shows the contents of the Startup script field (readonly). − Clear this check box to change the parameters passed to the script or to specify a different script: − To specify the script, click ... and select the desired script in the dialog that opens . − To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: − Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . − If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . − If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Specify the script to be used to stop the server. Use default: − If this check box is selected, the default script is used. (For certain JBoss versions, there is no default script and the server is stopped through its management API.) in this case opens the Default Shutdown Script dialog which shows the contents of the Shutdown script field (readonly). − Clear this check box to change the parameters passed to the script or to specify a different script: − To specify the script, click ... and select the desired script in the dialog that opens . − To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: − Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . − If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . − If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Pass environment variables	To pass specific variables to the server environment, select this check box and specify the variables: − To add a variable, click + and specify the variable name and value in the Name and Value fields respectively. − To remove a variable from the list, select the variable and click −.
Port	Use this field to change the debugger port.
Debugger Settings	Click this button to edit the debugger options on the Debugger page of the Settings dialog .

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click –. See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: Jetty Server

Run | Edit Configurations | + | Jetty Server | Local or Remote

Jetty Server run/debug configurations let you deploy and debug your applications on Jetty . (The Jetty Integration plugin must be enabled.)

- Name field and Share option
- Server tab for a local configuration
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .

JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
JMX port	The server JMX port.
Active	Select the check boxes to make the corresponding files active. For Jetty versions before 9, the list contains the server XML configuration files. For Jetty 9, the module `.mod` files are listed. The selected files are passed as command-line arguments to `<jetty_home>\start.jar` which is used to start the server.
Path	The paths to the corresponding Jetty configuration or module files are shown (readonly). For the files located within the Jetty installation directory, the paths are relative to the installation directory. For all the rest of the configuration files, the absolute paths are shown.
	Initially, the list contains the files located in `<jetty_home>\etc` or `<jetty_home>\modules`. Use the following icons to manage the list: Click this icon to add a file to the list. In the dialog that opens, select the necessary Jetty configuration or module file and click OK. Click this icon to remove the selected file from the list. Note that this operation does not delete the file physically. Click this icon to replace the selected file. In the dialog that opens, select the replacement file and click OK. and . Click the icon to move the selected file one line up or down in the list.

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action.
JMX Port	The server JMX port.
Remote staging	This section contains the settings related to staging. An example of remote staging settings for a mounted folder is provided after this table.
Type	Select the way the staging environment or host is accessed for transferring the application artifact or artifacts from your local computer. (In the user interface of IntelliJ IDEA this setting is also referred to as the connection type.) The available options are: – Same file system. Select this option if the target server is installed on your local computer. The artifacts in this case are deployed locally and, thus, don't need to be transferred to a remote host. – ftp. The File Transfer Protocol or Secure FTP is used. – Local or mounted folder. The staging environment is a local folder or is accessed as a mounted folder. If the list is empty, you have to enable the Remote Hosts Access plugin which supports the corresponding functionality.
Host	If Same file system is selected for Type, the only available option for Host is also Same file system. In all other cases, the list contains the existing configurations of the selected type. So each configuration corresponds to an individual (S)FTP connection, or a local or mounted folder. Select an existing configuration or create a new one. To create a new configuration: 1. Click to the right of the list. 2. In the Deployment dialog, click . 3. In the Add Server dialog, specify the configuration name, select the type, and click OK. 4. On the Connection tab, specify the settings in the Upload/download project files section. The rest of the settings don't matter. 5. Click OK in the Deployment dialog.
contexts	Location of one of the following directories: – For Jetty 9: `<jetty_home>\webapps` – For Jetty versions before 9: `<jetty_home>\contexts`

	(This is where IntelliJ IDEA generates the deployment descriptor XML file for the application artifact.)
Local path or Path from root	If Same file system is selected for Type and Host, specify the absolute path. In all other cases, specify the path relative to the local or mounted folder, or the root of the (S)FTP host. You can use … to select the folder in the Select Path or the Choose target path dialog.
Staging	When deploying to the remote host, the application artifact or artifacts are placed into a staging folder which should be accessible to Jetty. The settings in this section define the location of this staging folder. Note that if Same file system is selected for Type and Host, no settings in this section need to be specified.
Path from root	The path to the staging folder relative to the local or mounted folder, or the root of the (S)FTP host. You can use … to select the folder in the Choose target path dialog.
Mapped as	The absolute path to the staging folder in the local file system of the remote host.
Remote connection settings	The settings for accessing deployed applications. In Jetty run/debug configurations, these are used just to form the starting URL .
Host	The fully qualified domain name or the IP address of the Jetty host.
Port	The server HTTP port.

An example of remote staging settings for a mounted folder

Assuming that:

- `C:\shared` is a shared folder on the remote host which is mounted to the local computer as the drive `X:` .
- Jetty is installed in `C:\shared\jetty-distribution-9.2.7.v20150116` .
- The folder that you are going to use for staging is `C:\shared\staging` .

Here are the corresponding remote staging settings:

- Type: Local or mounted folder.
- Host: The configuration should be selected in which the value in the Folder field is `X:\` (the Upload/download project files section on the Connection tab of the Deployment dialog).
- contexts/Path from root: `jetty-distribution-9.2.7.v20150116\webapps` (For Jetty 8 or earlier version, that would be something like `jetty-distribution-8.0.1.v20110908\contexts` .)
- Staging/Path from root: `staging`
- Staging/Mapped as: `C:\shared\staging`

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. To edit the settings for an artifact or external resource, select the corresponding item in the list and use the controls in the right-hand part of the tab. For more information, see the table below.

Item	Description
+(Alt+Insert)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
–(Alt+Delete)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(F4)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)
Use custom context root	If you want to assign a particular context root to an artifact or external resource, select the artifact or the resource, select the check box and specify the desired context root in the field underneath the check box.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .

Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
−	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use or to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click −.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
−	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
−	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click −. See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: JSR45 Compatible Server

Use this dialog to create or edit run/debug configurations for JSR45-compatible application servers.

Generally, you may want to use this type of run/debug configuration under the following conditions:

- The server you are going to work with supports JSR-45 .
- IntelliJ IDEA doesn't provide a dedicated plugin for integration with this server.

On this page:

- Server tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab
- Before Launch options
- Toolbar

Server tab

Item	Local/Remote	Description
Application Server	Both	Select the server configuration to be used. If the run/debug configuration is intended for working with a remote server, the same server version should be configured locally and associated with the run/debug configuration. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
Start browser	Both	Select this check box to run the default Web browser to study your application output there.
With JavaScript debugger	Both	If this check box is selected, the Web browser is started with the JavaScript debugger enabled. JavaScript debugging is available for Firefox only. Note that when you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension will be installed.
Startup page	Both	In this field, specify the URL the browser should go to when started. In most typical cases, this URL will correspond to the root of your Web application or its starting page.
VM options	Local	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: - Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . - If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . - If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Local	Select the action to be performed when the application is updated in the Run or the Debug tool window ( ).
Show dialog	Local	Select this check box if you want to see the Update <application name> dialog every time you perform the Update action. The Update <application name> dialog is used to view and change the current update option (for example, Restart server) prior to actually updating the application.
JSP's package	Both	Specify the Java package prefix to be used for jsp-to-servlet translation.
VM options variable	Local	If there is a variable which stores the command-line JVM options, you can specify the name of this variable in this field. In this way, you can pass the corresponding options to the server JVM at the server start.
Port	Both	Specify the HTTP server port.
Use JSP's line mapping model specific for WebSphere 5.1	Both	If the run/debug configuration is intended for working with WebSphere Server 5.1, select this check box to be able to debug your JSPs. (To be able to debug JSPs, it's necessary to maintain relationships between the lines of the source files and the corresponding positions in the compiled code. All the servers except WebSphere 5.1 form such relationships similarly.)
Host	Remote	The DNS name or the IP address of the server host (for example, `localhost` , `127.0.0.1` , etc.).

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: - Full paths to specific files. - Ant patterns that define the range of files to be displayed. - Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .

Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
−	Click this button to remove the selected log entry from the list.
...	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use + or + to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click −.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab

Item	Local/Remote	Description
Run Debug	Both	Select Run or Debug to show settings either for the run or the debug mode.
Run with Coverage	Local	Select Run with Coverage to show settings for run with coverage mode.
Startup script	Local	Specify the script to be used to start the server. If necessary, you can also specify the script parameters and the options to be passed to the server JVM. You can provide all the necessary information right in the field, by typing. As an alternative: – To specify the script, click ... and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Local	Specify the script to be used to stop the server. If necessary, you can also specify the script parameters and the options to be passed to the server JVM. You can provide all the necessary information right in the field, by typing. As an alternative: – To specify the script, click ... and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Pass environment variables	Local	To pass specific variables to the server environment, select this check box and specify the variables: – To add a variable, click + and specify the variable name and value in the Name and Value fields respectively. – To remove a variable from the list, select the variable and click −.
Port	Local	Use this field to change the debugger port (if necessary).
Debugger Settings	Local	Click this button to edit the debugger options on the Debugger page of the Settings dialog .

Transport	Remote	Specify the "transport" settings for the connection with the debugger. In technical terms, these are the parameters for the `-Xrunjdwp` command-line option:
		– Socket. Specify the debugger port in the Port field. The combination of these two settings translates into
		`-Xrunjdwp:transport=dt_socket,address=<port>,suspend=n,server=y`
		– Shared memory. Specify the shared memory address in the corresponding field. The combination of these two settings translates into
		`-Xrunjdwp:transport=dt_shmem,address=<address>,suspend=n,server=y`
		Note that as you change the transport settings, what follows `transport=` within `-Xrunjdwp` in the area above also changes. In this way you control the corresponding command-line debugger parameters which you cannot edit directly.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click –. See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: JSTestDriver

In this dialog box, create a configuration to be used for running JavaScript unit tests in the browser against a JSTestDriver server. Configurations of this type enable running unit tests based on the JSTestDriver Assertion , Jasmine , and QUnit frameworks.

The dialog box is available when the JSTestDriver plugin is activated. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

On this page:

- Configuration Tab
  - Test
  - Server
- Debug Tab
- Coverage Tab
- Toolbar
- Common options

Configuration Tab

In this tab, specify the tests to run, the configuration files to use, and the server to run the tests against.

Test

In this area, tell IntelliJ IDEA where to find the tests to execute and how to get test runner configuration files that define which test files to load and in which order. The main approaches are:

- Specify the location of one or several previously created configuration files.
- Point at the target test file, test case, or test method, and then specify the location of the corresponding configuration file.

The way to find tests and configuration files is defined in the Test drop-down list. This choice determines the set of other controls in the area.

Item	Description	Available for
Test	In this drop-down list, specify how IntelliJ IDEA will get test runner configuration files. - All configuration files in directory: select this option to use all the test runner configuration files in a specific folder. - Configuration file: select this option to use a specific test runner configuration file. - JavaScript test file: select this option to have tests from a specific file executed using one of the previously defined configuration files. - Case: select this option to run a specific test case using one of the previously defined configuration files. - Method: select this option to run a specific test method using one of the previously defined configuration files.	
Directory	In this text box, specify the folder to look for test runner configuration files in. Type the path manually or click the Browse button ... and select the required folder in the dialog box, that opens.	All configuration files in directory
Matched configuration files	This read-only field shows a list of all the `*jstd` and `JSTestDriver.conf` test runner configuration files detected in the specified folder .	All configuration files in directory
Configuration file	In this text box, specify the test runner configuration file to use. Type the path manually or click the Browse button ... and select the required file in the dialog box that opens.	Configuration file
JS test file	In this text box, specify the JavaScript files with tests to be executed. Type the path manually or click the Browse button ... and select the required file in the dialog box, that opens.	JavaScript test file Case Method
Case	In this text box, type the name of the target case from the specified JavaScript file .	Case Method
Method	In this text box, type the name of the target method from the specified test case within the specified JavaScript file .	Method

Server

In this area, appoint the test server to run tests against.

Item	Description
At address	Choose this option to have the test execution handles by a remote test server. In the text box, specify the URL address to access the server through.
Running in IDE	Choose this option to have test execution handles through the JSTestDriver server that comes bundled with IntelliJ IDEA and can be launched from it.
Test Connection	Click this button to check that the specified test server is accessible. The server must be running. Start the server from IntelliJ IDEA or manually, according to the server-specific instructions.

Debug Tab

In this tab, appoint the browser to debug the unit test in when two or more browsers are captured simultaneously.

Item	Description
Debug	From this drop-down list, choose the browser to debug the specified tests in when two browsers are captured at a time. The available options are: - Chrome - Firefox

Coverage Tab

In this tab, specify the files that you do not want to be involved in coverage analysis.

Item	Description
Exclude file path	In this area, create a list of files to be skipped during coverage analysis. – To add an item to the list, click the Add button + and choose the required file in the dialog that opens . – To delete a file from the list so coverage is measured for it, select the file and click the Remove button –.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated.

In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .

- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: JUnit

JUnit run/debug configurations define how unit tests that are based on the JUnit testing framework should be run.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

It also contains a set of options that are common for all run/debug configurations and a toolbar that lets you manage your configurations.

Tip You can use `Ctrl+Space` to let IntelliJ IDEA help you fill in the fields in this dialog. (In the editor, `Ctrl+Space` is used for code completion .)

Configuration tab

Item	Description
Test kind	From this drop-down list, select the scope for your tests and fill in the fields depending on your selection.
All in package	Select this option to run all unit tests in the specified package. Fill in the following fields:
	Package — Specify package name
	Search for tests — Select where in your project IntelliJ IDEA shall look for test classes related to the current package: – In whole project: IntelliJ IDEA will look for test classes in all project modules – In single module: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field – Across multiple dependencies: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field, and in the modules that depend on it
All in directory	Select this option to run all unit tests in the specified directory. Fill in the following field:
	Directory — Specify the directory where you want to run the tests. It will act as the root directory for all relative input and output paths.
Pattern	Select this option to run a set of test classes. This set may include classes located in the same or different directories, packages or modules. Fill in the following fields:
	Pattern — Specify the required classes. Each class in this field must be represented by its fully qualified name. Class names must be separated with `||` . You can type class names manually, or click + on the right (or press `Shift+Enter`) and search for classes you want to add in the dialog that opens. You can also create a suite test, i.e. a bundle of several test classes that will be run together. To create a suite test class, click on the right and type the test classes you want to be run as a suite in the Configure suit tests dialog that opens. As a result, a new class will be created with the `@Suite` annotation.
	Method — Specify the method to be launched (passed to the JRE). Type method name, or click … and select the desired method in the dialog that opens.
	Search for tests — Select where in your project IntelliJ IDEA shall look for test classes related to the current package: – In whole project: IntelliJ IDEA will look for test classes in all project modules – In single module: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field – Across multiple dependencies: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field, and in the modules that depend on it
Class	Select this option to run all tests in a class. Fill in the following field:
	Class — Specify the fully qualified name of the class to be launched (passed to the JRE). Type the class name or click … and select the desired class in the dialog that opens.
Method	Select this option to run an individual test method. Fill in the following fields:
	Class — Specify the fully qualified name of the class to be launched (passed to the JRE). Type the class name or click … and select the desired class in the dialog that opens.
	Method — Specify the method to be launched (passed to the JRE). Type method name, or click … and select the desired method in the dialog that opens.
Category	Select this option if you only want to run test classes and test methods that are annotated either with the category given with the `@IncludeCategory` annotation, or a subtype of this category. Learn more about JUnit categories . Fill in the following fields:
	Category — Specify the desired category. Type category name, or click … and select the desired category in the dialog that opens.
	Search for tests — Select where in your project IntelliJ IDEA shall look for test classes related to the current package: – In whole project: IntelliJ IDEA will look for test classes in all project modules – In single module: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field – Across multiple dependencies: IntelliJ IDEA will look for test classes only in the module selected in the Use classpath of module field, and in the modules that depend on it
Fork mode	This option controls how many Java VMs will be created if you want to fork some tests. Select method or class to create a separate virtual machine for each method or class respectively. The available options in this drop-down list depend on the Test kind setting.
Repeat	If you want to repeatedly run a test, select the threshold from this drop-down list. You can select to run your test once, n times (in this case specify the number of times in the field on the right), until the test fails, or until it is stopped.
VM options	If necessary, specify the string to be passed to the VM. This string may contain the options such as `-mx` , `-verbose` , etc. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .

	If there is not enough space, you can click and enter the string in the dialog that opens. The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this field, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the required arguments in the dialog that opens. Use the same rules as for specifying the VM options .
Working directory	Specify the directory that will act as the current directory when running the test. It will act as the root directory for all relative input and output paths. By default, the directory where the project file resides, is used as a working directory. Type directory name, or click and select the desired directory in the dialog that opens. You can also click to switch between directories.
Environment variables	Click to open the Environment Variables dialog box where you can create variables and specify their values.
Use classpath of module	Select the module whose classpath should be used to run the tests.
JRE	Specify the JRE to be used. Select the JRE from the list, or click and select the installation folder of the required JRE in the dialog that opens.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description

+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: JUnit Test Discovery

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

It also contains a set of options that are common for all run/debug configurations and a toolbar that lets you manage your configurations.

Tip You can use Ctrl+Space to let IntelliJ IDEA help you fill in the fields in this dialog. (In the editor, Ctrl+Space is used for code completion .)

Configuration tab

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.

	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.

Alt+Up	Click this icon to move the selected task one line up in the list.
Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Karma

In this dialog box, create configurations for running and debugging JavaScript unit tests using the Karma test runner.

On this page:

– Getting Access to the Run/Debug Configuration: Karma Dialog Box
– Karma-Specific Configuration Settings
– Toolbar
– Common options

Getting Access to the Run/Debug Configuration: Karma Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS runtime environment that contains the Node Package Manager(npm) .
3. Using the Node Package Manager, install the Karma test runner as described in Preparing to Use Karma Test Runner .
4. Make sure the Karma plugin is activated. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Karma-Specific Configuration Settings

Item	Description
Name	In this text box, specify the name of the run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Node.js interpreter	
Karma Node.js package	In this field, specify the Karma installation home `/npm/node_modules/karma` . If you installed Karma regularly through the Node Package Manager, IntelliJ IDEA detects the Karma installation home itself. Alternatively, type the path to executable file manually, or click the Browse button and select the location in the dialog box, that opens.
Configuration file	In this field, specify the location of the Karma configuration file . Normally, the file has the extensions `.conf.js`

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the

CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Kotlin

The Kotlin run/debug configuration enables you running or debugging Kotlin applications.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

Click here for the description of the options that are common for all run/debug configurations.

Configuration tab

Item	Description
Main class	In this text box, specify the fully qualified name of the class to be executed (passed to the JRE). Type the class name manually or click the Browse button to open the Choose Main Class dialog box, where you can find the desired class by name or search through the project.
VM options	In this text box, specify the string to be passed to the VM for launching an application. Usually this string contains the options such as `-mx`, `-verbose`, etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options.
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens. Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. Note The list of environment variables may vary depending on the enabled plugins.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values. Note that you can copy-paste the contents of the Environment variables field without having to open the Environment Variables dialog box.
Use classpath of module	Select the module whose classpath should be used to run the application.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results.
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window.

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window.

Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts .

- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Kotlin-JavaScript (Experimental)

The Kotlin-JavaScript run/debug configuration enables you running or debugging Kotlin-JavaScript applications in the Ultimate edition of IntelliJ IDEA.

Kotlin-JavaScript options

Item	Description
Generated JavaScript file directory	In this text box, specify the directory, where the generated JavaScript file will be placed. You can type the path manually, or use the browse button ... to locate the directory in your file system.
Open in browser after translation	Select this check box to have the generated file opened in the selected browser. The following two fields become enabled only when this check box is selected.
HTML file	In this text box, specify the corresponding HTML file. You can type the file path manually, or use the browse button ... to locate the desired HTML file in your file system.
Browser	Select the desired browser from the drop-down list.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑ ↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.

Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.

- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Kotlin Script

The Kotlin script run/debug configuration enables you running or debugging Kotlin scripts.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab

Click here for the description of the options that are common for all run/debug configurations.

Configuration tab

Item	Description
Script file	Specify here the path to a `*.kts` file to be launched.
VM options	In this text box, specify the string to be passed to the VM for launching an application. Usually this string contains the options such as `-mx`, `-verbose`, etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options.
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens. Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. Note The list of environment variables may vary depending on the enabled plugins.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values. Note that you can copy-paste the contents of the Environment variables field without having to open the Environment Variables dialog box.
Use alternative JRE	Select the check box, if you want to specify an alternative JRE here; select the required JRE from the drop-down list, or click and choose the required JRE from the file chooser dialog.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results.
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.

or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription	
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut	
	Alt+Insert Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete Click this icon to remove the selected task from the list.
	Enter Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up Click this icon to move the selected task one line up in the list.
	Alt+Down Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Maven

This run/debug configuration is used to launch the Maven projects. The dialog box contains the following tabs:

- Parameters tab
- General tab
- Runner tab

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Item	Description
Name	In this text box, specify the name of the current run/debug configuration.

Parameters tab

Item	Description
Working directory	Specify the path to the Maven project file `pom.xml` .
Command line	Specify the Maven goals and options separated with space.
Profiles	Specify the profiles to be used separated with space.
Resolve Workspace Artifacts	We recommend that you use this check box if you have dependent modules in your project. By default, this check box is not selected. In this case, classes of dependent modules are searched in .jar files in a Maven local repository. If you select this check box, the classes of the dependent modules will be searched in the module compilation output. In this case, each time you make changes to the dependent module, you don't need to deploy them into a local repository.

General

Item	Description
Work offline	If this option is checked, Maven works in offline mode and uses only those resources that are available locally. This option corresponds to the `--offline` command line option.
Use plugin registry	Check this option to enable referring to the Maven Plugin Registry. This option corresponds to the `--no-plugin-registry` command line option.
Execute goals recursively	If this option is cleared, the build does not recur into the nested projects. Clearing this option equals to `--non-recursive` command line option.
Print exception stack traces	If this option is checked, exception stack traces are generated. This option corresponds to the `--errors` command line option.
Always update snapshots	Select this check box to always update snapshot dependencies.
Output level	Select the desired level of the output log, which allows plugins to create messages at levels of *debug*, *info*, *warn*, and *error*, or disable output log.
Checksum policy	Select the desired level of checksum matching while downloading artifacts. You can opt to fails downloading, when checksums do not match (`--strict-checksums`), or issue a warning (`--lax-checksums`).
Multiproject build fail policy	Specify how to treat a failure in a multiproject build. You can choose to: - Fail the build at the very first failure, which corresponds to the command line option `--fail-fast` . - Fail the build at the end, which corresponds to the command line option `--fail-at-end` . - Ignore failures, which corresponds to the command line option `--fail-never` .
Plugin update policy	Select plugin update policy from the drop-down list. You can opt to: - Check for updates, which corresponds to the command line option `--check-plugin-updates` . - Suppress checking for updates, which corresponds to the command line option `--no-plugin-updates` .
Threads `(-T option)`	Use this field to set the `-T` option for parallel builds. This option is available for Maven 3 and later versions. For more information, see parallel builds in Maven 3 feature.
Maven home directory	Use this drop-down list to select a bundled Maven version that is available (for Maven2, version 2.2.1 and for Maven3, version 3.0.5) or the result of resolved system variables such as `MAVEN_HOME` or `MAVEN2_HOME` . You can also specify your own Maven version that is installed on your machine. You can click … and select the necessary directory in the dialog that opens .
User settings file	Specify the file that contains user-specific configuration for Maven in the text field. If you need to specify another file, check the Override option, click the ellipsis button and select the desired file in the Select Maven Settings File dialog.
Local repository	By default, the field shows the path to the local directory under the user home that stores the downloads and contains the temporary build artifacts that you have not yet released. If you need to specify another directory, check the Override option, click the ellipsis button and select the desired path in the Select Maven Local Repository dialog.

Runner tab

Item	Description
VM Options	Specify the string passed to the VM for launching the Maven project. When specifying the options, follow these rules: - Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` .

- If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`.
- If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`.

JRE	Select JRE from the drop-down list.
Skip tests	If this option is checked, the tests will be skipped when running or debugging the Maven project.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values.
Properties	Specify the properties and their values to be passed to Maven.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window.

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window.
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: - Full paths to specific files. - Ant patterns that define the range of files to be displayed. - Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens.
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog.
−	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder. If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Meteor

Use this dialog box to create configurations for running and debugging Meteor applications.

The dialog box is available when the Meteor plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

On this page:

– Configuration Tab
– Browser / Live Edit Tab
– Toolbar
– Common options

Configuration Tab

Item	Description
Name	In this text box, specify the name of the current run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Meteor Executable	In this field, specify the location of the Meteor executable file (see Installing Meteor).
Program Arguments	In this field, specify the command line additional parameters to be passed to the executable file on start up, if applicable. These can be, for example, `--dev` , `--test` , or `--prod` to indicate the environment in which the application is running (development, test, or production environments) so different resources are loaded on start up.
Working Directory	In this field, specify the folder under which the application files to run are stored. This folder must contain a `.meteor` folder in the root to be treated a Meteor project. By default, the field shows the path to the IntelliJ IDEA project root. Technically, several Meteor projects that implement different applications can be combined within one single IntelliJ IDEA project. To run and debug these applications independently, create a separate run configuration for each of them with the relevant working directory. To avoid port conflicts, these run configurations should use different ports. In the Program Arguments field, specify a separate port for each run configuration in the format `--port=<port_number>` .
Environment Variables	In this field, specify the environment variables for the Meteor executable file, if applicable.

Browser / Live Edit Tab

In this tab, configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, IntelliJ IDEA creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated Meteor run configuration.

Item	Description
Open Browser	In the text box in this area, specify the URL address to open the application at. If you select the After Launch check box, the browser will open this page automatically after the application starts. Alternatively you can view the same result by opening the page with this URL address in the browser of your choice manually.
After Launch	Select this check box to have a browser started automatically after a debugging session is launched. Specify the browser to use in the drop-down list next to the check box. – To use the system default browser, choose Default. – To use a custom browser, choose it from the list. Note that Live Edit is fully supported only in Chrome. – To configure browsers, click the Browse button … and adjust the settings in the Web Browsers dialog box that opens. For more information, see Configuring Browsers .
With JavaScript Debugger	Select this check box to enable debugging the client-side code in the selected browser.

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
–	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Mocha

In this dialog box, create configurations for running and debugging JavaScript unit tests using the Mocha test framework.

On this page:

- Getting Access to the Run/Debug Configuration: Mocha Dialog Box
- Mocha-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: Mocha Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS runtime environment that contains the Node Package Manager(npm) .
3. Using the Node Package Manager, install the Mocha test framework as described in Preparing to Use Mocha Test Framework .

Mocha-Specific Configuration Settings

Item	Description
Node.js interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node options	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. See Node Parameters for details.
Working directory	In this field, specify the working directory of the application. All references in test scripts will be resolved relative to this folder, unless such references use full paths. By default, the field shows the project root folder. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button ... and select the location in the dialog box, that opens.
Environment variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button ... to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button + and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button –. – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.
Mocha package	In this field, specify the Mocha installation home `/npm/node_modules/mocha` . If you installed Mocha regularly through the Node Package Manager, IntelliJ IDEA detects the Mocha installation home itself. Alternatively, type the path to executable file manually, or click the Browse button ... and select the location in the dialog box, that opens.
User interface	From this drop-down list, choose the interface according to which the tests in the test folder are written. IntelliJ IDEA will recognize only tests that comply with the chosen interface. If during test execution IntelliJ IDEA runs against a test of another interface, the test session will be canceled with an error. This means that all the tests in the specified test folder must be written according to the same interface and this interface must be chosen from the drop-down list.
Tests	In this area, specify the tests to be executed. The available options are: – All in directory: choose this option to run all the tests from files stored in a folder. In the Test directory field, specify the folder with the tests. To have IntelliJ IDEA look for tests in the subfolders under the specified directory, select the Include subdirectories check box. – File pattern: choose this option to have IntelliJ IDEA look for tests in all the files with the names that match a certain mask and specify the mask in the Test file pattern field. – Test file: choose this option to run only the tests from one file and specify the path to this file in the Test file field. – Suite: choose this option to run individual suites from a test file. In the Test file field, specify the file where the required suites are defined. Click the Suite name field and configure a list of suites to run. To add a suite to the list, click + and type the name of the required suite. To remove a suite, select it in the list and click – – Test: choose this option to run individual tests from a test file. In the Test file field, specify the file where the required tests are defined. Click the Test name field and configure a list of tests to run. To add a test to the list, click + and type the name of the required test. To remove a test, select it in the list and click –

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
–	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: MXUnit

Use this dialog to create or edit MXUnit run/debug configurations which let you run or debug your MXUnit tests.

Click here for the description of the options that are common for all run/debug configurations.

Item	Description
Test	Select the test scope and specify the associated settings: – Directory. Select this option to run all tests in a certain directory. Specify the qualified directory name in the File field respectively. Alternatively, click … (Shift+Enter) and select the desired file in the Select Path dialog. – Method. Select this option to run a test method. Specify the fully qualified name of the test file and the method name in the File and Method fields respectively. Use … (Shift+Enter) to select the desired file and method in the Select Path and the Choose Test Method dialogs. – Component. Select this option to run a test for the specific component. Specify the fully qualified name of the test file in the File field respectively. Alternatively, click … (Shift+Enter) and select the desired file in the Select Path dialog.
Web Path	In this text box, specify the URL address that corresponds to the server root. The server side of deployment mappings will be specified relative to this URL.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.

- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Node JS

In this dialog box, create configurations for running and debugging of NodeJS applications locally. "Locally" in the current context means that IntelliJ IDEA itself starts the NodeJS runtime environment installed on your computer, whereupon initiates a running or debugging session.

On this page:

- Getting Access to the Run/Debug Configuration: NodeJS Dialog Box
- Configuration Tab
- Browser / Live Edit Tab
- V8 Profiling Tab
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: NodeJS Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS runtime environment that contains the Node Package Manager(npm) .

Configuration Tab

Item	Description
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the … button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node Parameters	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. The acceptable options are:

`--debug-brk==<port for connect to debugger remotely>` – With the `--debug-brk` option, the execution of the application suspends right after launch. This option allows you to debug the code executed on start. – With the `--debug` option, the code that has to be executed on the application start is executed whereupon the application waits for a debugger to connect to it. This option is useful when you are not going to debug Node.js right now, but you want to debug it later.	Specify this option to enable remote debugging of the application without re-starting the NodeJS server.
`-v, --version`	Print the current version of NodeJS.
`-e, --eval script`	Evaluate script.
`--v8-options`	Print v8 command line options.
`--vars`	Print various compiled-in variables.
`--max-stack-size=val`	Set max v8 stack size (bytes).
`--require coffee-script/register`	Specify this parameter to have CoffeeScript files transpiled into JavaScript on the fly during run. This mode requires that the `register.js` file, which is a part of the `coffee-script` package, should be located under the project root. Therefore you need to install the `coffee-script` package on the Node.js page locally, as described in Installing and Removing External Software Using Node Package Manager .

Working Directory	In this field, specify the working directory of the application. All references in the starting NodeJS application file, for example, includes, will be resolved relative to this folder, unless such references use full paths. By default, the field shows the project root folder. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button … and select the location in the dialog box, that opens.
JavaScript File	In this field, specify the full path to the file to start running or debugging the application from. If you are going to debug CoffeeScript, specify the path to the generated JavaScript file with source maps. The file can be generated externally or through transpilation using file watchers. For more details, see Transpiling CoffeeScript to JavaScript .
Application Parameters	In this text box, type the NodeJS-specific arguments to be passed to the application start file through the process.argv array.
Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button … to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button + and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button —. – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.
Path Mappings	

Browser / Live Edit Tab

In this tab, configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, IntelliJ IDEA creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated NodeJS run configuration.

However, activating Live Edit in this tab lets you invoke Live Edit without creating a separate configuration, during a debugging session launched through the Node.js configuration.

For more information, see Live Editing of HTML, CSS, and JavaScript .

Item	Description
Open Browser	In the text box in this area, specify the URL address to open the application at. If you select the After Launch check box, the browser will open this page automatically after the application starts. Alternatively you can view the same result by opening the page with this URL address in the browser of your choice manually.
After launch	Select this check box to have a browser started automatically after a debugging session is launched. Specify the browser to use in the drop-down list next to the check box. – To use the system default browser, choose Default. – To use a custom browser, choose it from the list. Note that Live Edit is fully supported only in Chrome. – To configure browsers, click the Browse button … and adjust the settings in the Web Browsers dialog box that opens. For more information, see Configuring Browsers .
With JavaScript debugger	Select this check box to enable debugging the client-side code in the selected browser.

V8 Profiling Tab

Item	Description
Record CPU profiling info	Select this check box to start logging the CPU profiling data when the application is launched. The controls in the area below become enabled. Specify the following: – Log folder: in this field, specify the folder to store recorded logs in. Profiling data are stored in V8 log files `isolate-<session number>` . – One log file for isolates: – Tick package: in this field, specify the tick package to use. Choose the relevant package from the Tick package drop-down list or click the … button next to it and choose the package in the dialog box that opens. – Gnuplot package: in this field, specify the location of the Gnuplot executable file to explore a timeline view that shows where V8 is spending time.
Allow taking heap snapshots	Select this check box to... The controls in the area below become enabled. Specify the following: – V8 profiler package: in this field, specify the v8-profiler package to use. Choose the relevant package from the v8-profiler package drop-down list or click the … button next to it and choose the package in the dialog box that opens. – Communication port: in this field, specify the port through which IntelliJ IDEA communicates with the profiler, namely, sends a command to take a snapshot when you click the Take Heap Snapshot button on the toolbar of the Run tool window.

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
–	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Node.js Interpreters Dialog

File | Settings | Languages and Frameworks | Node.js and NPM for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | Node.js and NPM for OS X

The dialog box opens when you click the Browse button next to the Node Interpreter drop-down list in the Run/Debug Configuration: Node JS dialog or on the Node.js and NPM page.

Use this dialog box to configure Node.js installations as local and remote interpreters.

The term local Node.js interpreter denotes a Node.js installation on your computer. The term remote Node.js interpreter denotes a Node.js installation on a remote host or in a virtual environment set up in a Vagrant instance. See Configuring Node.js Interpreters for details.

When the dialog box opens from the Node.js and NPM page, you can only configure local interpreters installed on your computer. When the dialog box is accessed from the Run/Debug Configuration: Node JS dialog, both local and remote interpreters can be configured.

ItemTooltipDescription		
Node.js Interpreters		The list shows all the configured Node.js interpreters, both local and remote ones. For local interpreters, IntelliJ IDEA also shows the path to the Node.js executable file and to the associated npm executable file, see Installing and Removing External Software Using Node Package Manager .
	Add	Click this button to add a new Node.js interpreter to the list. From the drop-down menu, choose All Local or Add Remote. Note that the Add Remote item is available only you opened the dialog box from the Run/Debug Configuration: Node JS dialog. Depending on your choice, either select the relevant local Node.js installation or configure a remote interpreter in the Configure Node.js Remote Interpreter Dialog that opens.
	Delete	Click this button to remove the selected interpreter from the list.
	Edit	Click this button to create a new interpreter with the settings copied from the selected one.

Configure Node.js Remote Interpreter Dialog

The dialog box is available only when the Node.js Remote Interpreter plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The dialog box opens when you click the Add toolbar button + in the Node.js Interpreters Dialog and choose Remote... from the drop-down menu. This menu item is available only when you open the Node.js Interpreters Dialog from the Run/Debug Configuration: Node JS .

Use this dialog box to configure access to Node.js installations on remote hosts or in development environments set up in Vagrant instances.

Item	Description

SSH Credentials	Choose this option to configure access to a Node.js interpreter on a remote host through SSH credentials. In the fields of the dialog box, specify the following:
	– Host: in this field, type the name of the host on which the interpreter is installed.
	– Port: in this field, type the port which the SSH server on the remote host listens to. The default port number is 22.
	– User name: in the field, type the user name under which you are registered on the SSH server.
	– Auth type: from this drop-down list, choose the authentication method.
	  – To access the host through a password, choose Password from the Auth type drop-down list and type the password.
	  – To access the host through a pair of SSH keys, choose Key pair, then specify the path to the file where your private key is stored or the passphrase if you have configured it during the generation of the key pair.
	
	To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . IntelliJ IDEA first attempts to retrieve path mappings itself by processing all the available application-level configurations. If IntelliJ IDEA finds the configurations with the same host as the one specified above, in the Host field, the mappings from these configurations are merged automatically. If no configurations with this host are found, IntelliJ IDEA displays an error message informing you that path mappings are note configured.
	
	To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the server access configuration in question, switch to the Mappings tab, and map local folders to folders on the server as described in Creating a Remote Server Configuration , section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.

Vagrant	This option is available only when the Vagrant repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Choose this option to configure access to a Node.js interpreter installed in a Vagrant instance using your Vagrant credentials. Technically, it is the folder where the VagrantFile configuration file for the desired environment is located. Based on this setting, IntelliJ IDEA detects the Vagrant host and shows it as a link in the Vagrant Host URL read-only field. To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . IntelliJ IDEA evaluates path mappings from the `VagrantFile` configuration file.
Deployment Configuration	This option is available only when the Remote Hosts Access plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins . Choose this option to configure access to a Node.js interpreter on a remote host using a server access configuration. This option is available only if you have at least one server access configuration of the type SFTP, see Creating a Remote Server Configuration . From the Deployment Configuration drop-down list, choose the server access configuration of the type SFTP according to which you want IntelliJ IDEA to connect to the target host. If the settings specified in the chosen configuration ensure successful connection, IntelliJ IDEA displays the URL address of the target host as a link in the Deployment Host URL field. To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . By default, IntelliJ IDEA retrieves path mappings from the chosen server access (deployment) configuration. If the configuration does not contain path mappings, IntelliJ IDEA displays the corresponding error message. To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the relevant server access configuration, switch to the Mappings tab, and map the local folders to the folders on the server as described in Creating a Remote Server Configuration , section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.
Node.js Interpreter Path	In this field, specify the location of the Node.js executable file in accordance with the development environment configured on the selected remote host. By default IntelliJ IDEA suggests the `/usr/bin/node` folder. To specify another folder, click the Browse button ... and choose the relevant folder in the dialog box that opens. Note that the Node.js home directory must be open for edit. When you click OK, IntelliJ IDEA checks whether the Node.js executable is actually stored in the specified folder. – If no Node.js executable is found, IntelliJ IDEA displays an error message asking you whether to continue searching or save the interpreter configuration anyway. – If the Node.js executable is found, you return to the Node.js Interpreters where the installation folder and the detected version of the Node.js interpreter are displayed.

Run/Debug Configuration: Node JS Remote Debug

In this dialog box, create configurations for debugging already running Node.js applications. This approach gives you the possibility to re-start a debugging session without re-starting the NodeJS server.

On this page:

- Getting Access to the Run/Debug Configuration: NodeJS Remote Debug Dialog Box
- NodeJS Remote Debug-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: NodeJS Remote Debug Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS runtime environment that contains the Node Package Manager(npm) .

NodeJS Remote Debug-Specific Configuration Settings

Item	Description
Host	In this text box, specify the host where the application is running.
Debug port	In this text box, specify the port to connect to. Do one of the following: – Copy the port number from the information message in the Run tool window that controls the running application. – Copy the port number from the Node parameter text box in the Run/Debug Configuration:NodeJS dialog box.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.

See also, Working with Artifacts .

- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: NodeUnit

In this dialog box, create configurations to run unit tests for NodeJS applications.

On this page:

- Getting Access to the Run/Debug Configuration: NodeUnit Dialog Box
- NodeUnit-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: NodeUnit Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS runtime environment that contains the Node Package Manager(npm) .
3. Install the Nodeunit testing framework in one of the following ways:
   - Download the framework at https://github.com/caolan/nodeunit and install it according to the official instructions.
   - Install the `nodeunit` package using the Node Package Manager (NPM) as described in Installing and Removing External Software Using Node Package Manager .

NodeUnit-Specific Configuration Settings

Item	Description
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the Browse button and select the location in the dialog box, that opens.
Working Directory	In this text box, specify the folder to find tests under. This can be the project root folder or the parent directory for the `test` folder. Type the path manually or click the Browse button and select the location in the dialog box, that opens.
Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button . – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.
Nodeunit Module	In this text box, specify the installation folder of the Nodeunit framework. Type the path manually or click the button and choose the folder in the dialog box that opens.
Run	From this drop-down list, choose the scope of tests to execute. The available options are: – All JavaScript test files in the directory: choose this option to to have IntelliJ IDEA run all the test files in a folder. In the Directory text box below, specify the path to the test folder relative to the working directory . – JavaScript test file: choose this option to have a specific test executed. In the JavaScript test file text box, type the path to the file relative to the working directory .

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Node-webkit

In this dialog box, create configurations for running and debugging JNode-webkit applications.

On this page:

– Node-Webkit-Specific Configuration Settings

– Toolbar

– Common options

Node-Webkit-Specific Configuration Settings

Item	Description
Node-webkit app	Specify the application to be launched. You can either select the entire directory that contains `package.json` file, or the same directory packed into an archive with the `.nw` extension. Use the following buttons: `Shift+Enter` Click this button to find the desired application in the file system. Click this button to add macros, specified in the Path Variables page of the IDE settings.
Node-webkit arguments	Specify here the arguments to be passed to the application being launched. Use the following button: `Shift+Enter` Click this button to open the textual editor.
Working directory	In this field, specify the working directory of the application. All references in test scripts will be resolved relative to this folder, unless such references use full paths. By default, the field shows the project root folder. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button and select the location in the dialog box, that opens.
Environment variables	In this field, specify the environment variables for the executable file, if applicable.
Node-webkit interpreter	Specify executable for the Node-webkit. Use the following buttons: `Shift+Enter` Click this button to find the desired application in the file system. Click this button to add macros, specified in the Path Variables page of the IDE settings.

Toolbar

	Item	Shortcut	Description
		`Alt+Insert`	Click this button to add a new configuration to the list.
		`Alt+Delete`	Click this button to remove the selected configuration from the list.
		`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults		Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or		`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder		Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations		Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: NPM

In this dialog box, create configurations for running npm scripts locally. "Locally" in the current context means that IntelliJ IDEA itself starts the NodeJS runtime environment installed on your computer, whereupon initiates script execution.

On this page:

- Getting Access to the Run/Debug Configuration: NPM Dialog Box
- Configuration
  - Toolbar
  - Common options

Getting Access to the Run/Debug Configuration: NPM Dialog Box

1. Install and enable the Node.js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
2. Download and install the NodeJS framework that contains the Node Package Manager(npm) .

Configuration

Item	Description
Name	In this text box, specify the name of the run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
package.json	In this field, specify the `package.json` file to run the scripts from. Choose the file from the drop-down list which shows all the `package.json` files detected in the current project or click … and choose the required `package.json` in the dialog box that opens.
Command	From this drop-down list, choose the npm CLI command to execute, by default `run-script` is selected. Learn more at npm documentation , under the section CLI Commands.
Scripts	From this drop-down list, choose the script to which the chosen command will be applied. The list contains all the scripts defined within the `scripts` property in the `Package.json` file.
Arguments	In this field, specify the command line arguments to execute a script with. Learn more at Arguments .
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the … button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node Options	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. The acceptable options are:

`--debug-brk==<port for connect to debugger remotely>` – With the `--debug-brk` option, the execution of the application suspends right after launch. This option allows you to debug the code executed on start. – With the `--debug` option, the code that has to be executed on the application start is executed whereupon the application waits for a debugger to connect to it. This option is useful when you are not going to debug Node.js right now, but you want to debug it later.	Specify this option to enable remote debugging of the application without re-starting the NodeJS server.
`-v, --version`	Print the current version of NodeJS.
`-e, --eval script`	Evaluate script.
`--v8-options`	Print v8 command line options.
`--vars`	Print various compiled-in variables.
`--max-stack-size=val`	Set max v8 stack size (bytes).
`--require coffee-script/register`	Specify this parameter to have CoffeeScript files transpiled into JavaScript on the fly during run. This mode requires that the `register.js` file, which is a part of the `coffee-script` package, should be located under the project root. Therefore you need to install the `coffee-script` package on the Node.js page locally, as described in Installing and Removing External Software Using Node Package Manager .

Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button … to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button + and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button —. – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
−	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.

	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: OpenShift Deployment

Run | Edit Configurations | + | OpenShift Deployment

OpenShift Deployment run/debug configurations let you deploy your code and application artifacts to OpenShift. They also let you debug your applications.

For OpenShift Deployment run/debug configurations to be available, the OpenShift integration plugin must be enabled.

- Main settings
- Before Launch options
- Toolbar

Main settings

Item	Description
Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in `.idea/workspace.xml` or the `.iws` file. If the check box is selected, the settings are stored in a separate `.xml` file in `.idea/runConfigurations` or in the `.ipr` file. See Project format and files.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.
Server	Select the cloud access configuration to be used. To create a new configuration, or to edit an existing one, click … (Shift+Enter). For more information, see OpenShift.
Deployment	To deploy your source code, select the corresponding module . To deploy an application artifact , select the artifact. Only archive artifact formats can be used (e.g. WAR, EAR).
Use custom application name	By default, your application will have about the same name as the module or the artifact. To specify a different name, select the check box and specify the name in the field. (The application name defines its URL.)
Debug Port	The port to be used for debugging. This may be any unused port on your computer.

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Debug tool window opens when you start the run/debug configuration in the debug mode. Otherwise, the tool window isn't shown. However, when the configuration is running in the debug mode, you can open the Debug tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.

You can group run/debug configurations by placing them into folders.
To create a folder, select the configurations to be grouped and click . Specify the name of the folder.

Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder.

To remove grouping, select a folder and click .

See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: OSGi Bundles

Use this dialog box to set up options for running and debugging applications that use OSGi Bundles .

The dialog box consists of the following tabs:

- Framework & Bundles
- Parameters
- Additional Framework Properties

Click here for the description of the options that are common for all run/debug configurations.

Framework & Bundles Tab

In this tab, select a framework that you need, compose a list of bundles to be installed, specify whether each bundle should be started upon installation, and define the order in which you want the bundles to be started.

Item	Description
OSGi Framework	From a drop-down list choose your framework.
Framework Start Level	Use this field to define a state of the execution in which a framework exists. You can modify the start level of the framework. This start level is used for managing the order of OSGi bundles' execution. If the bundle has a start level grater than the one for the framework, then it will be executed first.
Default Start Level	This field shows the default start level of bundles. In this case, the bundles are added after the framework's execution has started. You can modify the default start level for bundles.
Bundles Name	This read-only field shows the names of the OSGi bundles to be installed.
Start Level	In this text box, specify the default start level for newly installed bundles and thus determine the start order of bundles. The default value is 1. **Note** An OSGi system has a current level (called the active start level). If a bundle has a start level higher than the active start level it will not start when the OSGi system starts. The bundle will start as soon as the active start level reaches or exceeds the start level of the bundle. Accordingly, if the active start level becomes below the level of a bundle, the bundle will be shutdown.
Start After Install	Select this check box to have the selected bundle after installation.
Add	Click this button to open the Select Bundles dialog box, which displays all the currently available bundles. If the list is large, start typing the bundle name Search field - the contents of the list change as you type and show only the matching entries.
Remove	Click this button to delete the selected bundle from the list.

Parameters Tab

In these tab, customize the framework run or debug procedure by specifying additional parameters.

Item	Description
VM Options	In this text box, specify the string to be passed to the Virtual Machine for launching the bundles. If the string is too long and does not fit in the text box, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: - Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . - If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . - If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . **Note** The `-classpath` option specified in this field overrides the classpath of the module.
Program Parameters	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If the string is too long and does not fit in the text box, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options .
JRE	By default, the project JDK is used to run the bundles. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Runtime Directory	Use this area to specify the runtime path of the framework. You can select from the following options: - Recreate each time - select this option if you want to have all previous information deleted from the directory before the execution of the framework. - Use this directory - use this field to specify the directory that the framework will use each time it executes. The execution time is faster. However, the directory might contain the unnecessary information such as artifacts from previous runs. Type the path to the desired folder manually or click the Browse button and choose the folder in the dialog that opens .
Include All Bundles in Class Path	Select this check box to have all the selected bundles included in the classpath.

Additional Framework Properties

Item	Description
Debug Mode	Select this check box to enable debugging.
System Packages	In this text box, specify the system packages to be exposed inside the OSGi framework. Type the names of the packages using commas as separators. Wildcards are welcome.

Boot Delegation	In this text box, specify `java` packages for which the framework must delegate class loading to the boot class path. Type the names of the packages using commas as separators. Wildcards are welcome.
Start OSGi Console	Select this check box to run a prompt for the specified framework. For example, for the Equinox framework it is `osgi>` .
Run	Use this area to indicate what target you need to run. You can select from the following options: – Just the bundles - select this option if you need to run just the bundles. – Product - select this option if you need to run the product. – Application - select this option if you need to run the application.

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
−	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑ ↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it,

and specify the arguments to execute the script with.
Specify the location of the Node.js interpreter and the parameters to pass to it.

- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PhoneGap/Cordova

Use this dialog box to create configurations for running and debugging applications developed through integration with the help of PhoneGap , Apache Cordova , and Ionic frameworks and intended for running on various mobile platforms, including Android .

On this page:

- Getting Access to the Run/Debug Configuration: PhoneGap/Cordova Dialog Box
- PhoneGap/Cordova/Ionic-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: PhoneGap/Cordova Dialog Box

Make sure the PhoneGap/Cordova plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

PhoneGap/Cordova/Ionic-Specific Configuration Settings

Item	Description
PhoneGap/Cordova Executable Path	In this field, specify the location of the executable file `phonegap.cmd` , `cordova.cmd` , or `ionic.cml` (see Installing PhoneGap/Cordova/Ionic).
PhoneGap/Cordova Working Directory	In this field, specify the folder under which the PhoneGap/Cordova/Ionic application files to run are stored.
Command	From this drop-down list, choose the command to run. The contents of the drop-down list, depend on the actually used framework, namely, on the executable file specified in the PhoneGap/Cordova Executable Path field. The available options are: – For PhoneGap: – emulate – run – prepare – serve – remote build – remote run See https://www.npmjs.org/package/phonegap for a list of PhoneGap-specific commands with descriptions. – For Cordova: – emulate – run – prepare – serve See https://www.npmjs.org/package/cordova for a list of Cordova-specific commands with descriptions. – For Ionic: – emulate – run – prepare – serve See https://www.npmjs.org/package/ionic for a list of Ionic-specific commands with descriptions.
Platform	From this drop-down list, choose the platform for running on which the application is intended. The available options are: – Android – ios To emulate this platform, you need to install the ios-sim command line tool globally. You can do it through the Node Package Manager (npm), see Installing and Removing External Software Using Node Package Manager or by running the `sudo npm install ios-sim -g` command, depending on your oeprating system. – amazon-fireos – firefoxos – blackberry10 – ubuntu – wp8 – windows8 – browser Learn more about targeted platforms at http://docs.phonegap.com/en/edge/guide_platforms_index.md.html#Platform%20Guides and http://cordova.apache.org/docs/en/4.0.0/guide_cli_index.md.html#The%20Command-Line%20Interface .
Specify Target	Select this check box to appoint an Android physical or virtual device to run the application on and select the required device from the drop-down list. The list shows all the virtual and physical devices that are currently configured on our machine. See http://docs.phonegap.com/en/edge/guide_platforms_android_index.md.html#Android%20Platform%20Guide for details.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly

		created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ⬆⬇ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
⬆	Alt+Up	Click this icon to move the selected task one line up in the list.
⬇	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PHP Built-in Web Server

Use this dialog box to configure running PHP applications on a local built-in Web server.

Click here for the description of the options that are common for all run/debug configurations.

The dialog box consists of the following areas:

- Server Configuration
- Command Line
- Toolbar
- Common options

Server Configuration

In this area, configure the access to the built-in Web server.

Item	Description
Host	In this text box, type `localhost`, because the built-in server is located on your machine.
Port	Use this spin box to specify the number of the port where the built-in web server runs. The port number should be set to 1024 or higher. By default this port is set to the default IntelliJ IDEA port through which IntelliJ IDEA accepts connections from services (`63342`).
Document root	In this text box, type the full path to the folder that will be considered server document root . This can be either the project root or any other folder under it. All the folders under the document root will be recursively submitted to the PHP interpreter. The server document root folder will be accessed through HTTP at the above specified host:port. URL addresses for other pages of your applications will be composed based on this mapping. Type the path manually or click the Browse button and choose the relevant folder in the dialog box, that opens.
Use router script	Select this check box to have a PHP router script executed at every HTTP request start-up. The script is run for each HTTP request. If this script returns `FALSE`, the requested resource is returned as-is. Otherwise the script output is returned to the browser. In the text box, specify the location of the script to run. Type the path manually or click the Browse button and choose the file in the dialog box, that opens.

Command Line

In this area, customize the behavior of the current PHP interpreter by specifying the options and arguments to be passed to the PHP executable file.

Item	Description
Interpreter options	In this text box, specify the options to be passed to the PHP executable file. They override the default behavior of the PHP interpreter and/or ensure that additional activities are performed. If necessary, click and type the desired options in the Command Line Options dialog box. Type each option on a new line. When you close the dialog box, they are all displayed in the Command line options text box with spaces as separators.
Custom working directory	In this text box, specify the location of the files that are outside the folder with your sources and are referenced through relative paths. Type the path manually or click the Browse button and select the desired folder in the dialog that opens .
Environment variables	In this field, specify the environment variables be passed to the built-in server. See Environment Variables in Apache for details.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the

run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise.

If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PHP HTTP Request

Use this dialog box to configure running and debugging of separate HTTP Requests . This is helpful when you are actually interested in a specific page that is accessed in a number of steps, but for this or that reason you cannot specify this page as the start page for debugging, for example, because you need to "come" to this page with certain data. For details, see Debugging a PHP HTTP Request .

The dialog box consists of the following areas:

- Configuration
- Toolbar
- Common options

Click here for the description of the options that are common for all run/debug configurations.

Configuration

Item	Description
Server	From this drop-down list, select the server access configuration to interact with the Web server where the application is executed.
URL	In this text box, specify the `host` element of the request in question. Type the element relative to the host specified in the debug server configuration, see Creating a PHP Debug Server Configuration . As you type, IntelliJ IDEA composes the URL address on-the-fly and displays it below the text box.
Request method	From this drop-down list, choose the relevant request type. The available options are: - POST - GET
Query String	In this text box, type the query string of the request, this string will be appended to the request after the `?` symbol.
Request Body	In this text box, type the data to be sent to the server through the POST request. The text box is available only for request methods of the type POST. By default, the Project Encoding is used in requests encoding if it is not specified explicitly, for example: `header('Content-type: text/html;charset=utf-8');` The Project Encoding is specified on the File Encodings page, under the Editor node of the Settings / Preferences Dialog .

Toolbar

	Item	Shortcut	Description
+		Alt+Insert	Click this button to add a new configuration to the list.
−		Alt+Delete	Click this button to remove the selected configuration from the list.
		Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults		Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓		Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder		Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations		Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list. Item Keyboard Description

	shortcut	
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PHP Remote Debug

Use this dialog box to configure debugging of PHP applications on a remote server.

The dialog box consists of the following areas:

- Configuration
- Toolbar
- Common options

Click here for the description of the options that are common for all run/debug configurations.

Configuration

Item	Description
Server	Use this drop-down list to specify the Web server configuration to use. The list shows all the configurations that are currently available in IntelliJ IDEA.
…	Click this button to open the Deployment page and view the details of the selected configuration there.
Ide key (session id)	In this text box, specify the key to identify the debugging session.
Debug	Use the controls in this area to configure behaviour of the debugging tool. – Break at the first line - select this check box to have the debugging tool stop at the first line of the source code.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .

- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files .

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PHP Script

Use this dialog box to configure running and debugging of single PHP files locally using a PHP console.

Click here for the description of the options that are common for all run/debug configurations.

The dialog box consists of the following areas:

- Configuration
- Command line
- Debug
- Toolbar
- Common options

Configuration

In this area, specify the script to run or debug and the parameters to process it with, if applicable.

Item	Description
File	In this text box, specify the location of the file to run or debug. Type the path to the file manually or click the Browse button and select the desired location in the Choose PHP File dialog box that opens.
Arguments	In this text box, type the list of arguments to be passed to the PHP script, same way as if you were entering these parameters in the command line. If necessary, click and type the desired switches in the Arguments dialog box. Type each argument on a new line. As you type, they appear in the Arguments text box with spaces as separators.

Command line

In this area, customize the behavior of the current PHP interpreter by specifying the options and arguments to be passed to the PHP executable file.

Item	Description
Interpreter options	In this text box, specify the options to be passed to the PHP executable file. They override the default behavior of the PHP interpreter and/or ensure that additional activities are performed. If necessary, click and type the desired options in the Command Line Options dialog box. Type each option on a new line. When you close the dialog box, they are all displayed in the Command line options text box with spaces as separators.
Custom working directory	In this text box, specify the location of the files that are outside the folder with the script and are referenced in your script through relative paths. Type the path manually or click the Browse button and select the desired folder in the dialog that opens . This setting does not block the script execution because the script location is always specified through a full path.
Environment variables	In this field, specify the environment variables be passed to the built-in server. See Environment Variables in Apache for details.

Debug

Use the controls in this area to configure behaviour of the debugging tool.

Item	Description
Break at the first line	Select this check box to have the debugging tool stop at the first line of the source code.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.

Share

Select this check box to make the run/debug configuration available to other team members.
If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise.

If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PHP Web Application

Use this dialog box to configure running and debugging of PHP applications on a remote server. According to the settings of this configuration, IntelliJ IDEA fully controls the debugging process: it launches the application, opens the browser, and activates the debugging engine. A PHP Web Application debug configuration tells IntelliJ IDEA the URL address to access the starting page of the application, the browser to open the starting page in, and the debug server configuration to use. For details, see Debugging with a PHP Web Application Debug Configuration .

Click here for the description of the options that are common for all run/debug configurations.

The dialog box consists of the following areas:

- Configuration
- Toolbar
- Common options

Configuration

Item	Description
Server	Use this drop-down list to specify the Web server configuration to use. The list shows all the configurations that are currently available in IntelliJ IDEA.
	Click this button to open the Servers page and view the details of the selected configuration there.
Start URL	In this area, compose the URL address to access the application through. In the Start URL text box, specify the local file that implements the starting page of the application. Type the path to the desired file relative to the folder that is mapped to the root folder of the target host. IntelliJ IDEA concatenates the host root URL with the specified relative path and shows the URL address of the application starting page in the read-only field below.
Browser	From this drop-down list, select the Web browser to open the application in.
Debug	Use the controls in this area to configure behaviour of the debugging tool. – Break at the first line - select this check box to have the debugging tool stop at the first line of the source code.

Toolbar

Item	Shortcut	Description
	Alt+Insert	Click this button to add a new configuration to the list.
	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools .

- Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files .

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PHPUnit

Use this dialog box to create a configuration to be used for running and debugging unit tests on PHP applications in the console using the PHPUnit framework. Click here for the description of the options that are common for all run/debug configurations.

On this page:

- Before You Start
- Test Runner Area
- Command Line Area
- Toolbar
- Common options

Before You Start

To run PHPUnit tests:

1. Install and configure the PHPUnit tool on your computer, see http://www.phpunit.de/manual/current/en/installation.html .
2. Make sure the PHP plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Test Runner Area

In this area, specify the unit tests to launch and the command line switches to be passed to the test runner.

Item	Description
Test scope	In this area, specify the location of tests or the configuration file where they are listed. – Directory: select this option to have all the unit tests in a directory launched. In the Directory text box, specify the directory to search the unit test in. Type the path to the directory manually or click the Browse button and select the desired directory in the Choose Test Directory dialog box, that opens. – Class: select this option to have all the unit tests in a test class launched. 1. In the File text box, specify the file to search the class in. Type the path to the file manually or click the Browse button and select the desired directory in the Choose Test File dialog box, that opens. 2. In the Class text box, specify the desired class. Type the class name manually or click the Browse button and select the desired class in the tree view, that opens. – Method: select this option to have a specific test method launched. 1. In the File text box, specify the file to search for the test method in. Type the file name manually or click the Browse button and select the desired file in the tree view, that opens. 2. In the Method text box, specify the desired method. – Defined in the configuration file: select this option to have test runner execute the tests from a dedicated XML file. – To use the default configuration file specified on the PHPUnit page of the Settings dialog box, clear the Use alternative configuration file check box. If no default configuration file is appointed on the PHPUnit page, the run/debug configuration is invalid. – To run the tests from a custom configuration file, select the Use alternative configuration file check box and specify the location of the file to use in the text box.
Test runner options	In this text box, specify the test runner switches . If necessary, click and type the desired switches in the Command Line Options dialog box. Type each switch on a new line. When you close the dialog box, the specified switches are displayed in the Test runner options text box with spaces as separators.

Command Line Area

In this area, customize the behavior of the current PHP interpreter by specifying the options and arguments to be passed to the PHP executable file.

Item	Description
Interpreter options	In this text box, specify the options to be passed to the PHP executable file. They override the default behavior of the PHP interpreter and/or ensure that additional activities are performed. If necessary, click and type the desired options in the Command Line Options dialog box. Type each option on a new line. When you close the dialog box, they are all displayed in the Command line options text box with spaces as separators.
Custom working directory	In this text box, specify the location of the files that are outside the folder with tests and are referenced in your tests through relative paths. Type the path manually or click the Browse button and select the desired folder in the dialog that opens . This settings does not block the test execution because the location of tests is always specified through a full path to the corresponding files and/or directories.
Environment variables	In this field, specify the environment variables be passed to the built-in server. See Environment Variables in Apache for details.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.

	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PHPUnit by HTTP

Use this dialog box to configure running unit tests of PHP applications on a remote server./p> Click here for the description of the options that are common for all run/debug configurations.

On this page:

- Before You Start
- Configuration Tab
- Test Groups Tab
- Remote Tab
- Toolbar
- Common options

Before You Start

To run PHPUnit tests:

1. Install and configure the PHPUnit tool on your computer, see http://www.phpunit.de/manual/current/en/installation.html .
2. Make sure the PHP plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Configuration Tab

In this tab, specify the unit tests to launch and the command line switches to be passed to the test runner.

Item	Description
Test	In this area, specify the location of tests or the configuration file where they are listed. – All in Directory: select this option to have all the unit tests in a directory launched. In the Directory text box, specify the directory to search the unit test in. Type the path to the directory manually or click the Browse button … and select the desired directory in the Choose Test Directory dialog box, that opens. – All in File: select this option to have all the unit tests in a test class or a file launched. In the File text box, specify the file to search the class or suite in. Type the path to the file manually or click the Browse button … and select the desired directory in the Choose Test File dialog box, that opens. – Class or Suite: select this option to have all the unit tests in a test class or a test suite launched. 1. In the File text box, specify the file to search the class or suite in. Type the path to the file manually or click the Browse button … and select the desired directory in the Choose Test File dialog box, that opens. 2. In the Class text box, specify the desired class. Type the class name manually or click the Browse button … and select the desired class in the tree view, that opens. – Method: select this option to have a specific test method launched. 1. In the File text box, specify the file to search for the test method in. Type the file name manually or click the Browse button … and select the desired file in the tree view, that opens. 2. In the Method text box, specify the desired method. – XML file: select this option to have the test runner execute the tests from an XML configuration file. – To use the default configuration file specified on the PHPUnit page of the Settings dialog box, clear the Use alternative configuration file check box. If no default configuration file is appointed on the PHPUnit page, the run/debug configuration is invalid. – To run the tests from a custom configuration file, select the Use alternative configuration file check box and specify the location of the file to use in the text box.
Test Runner options	
Custom working directory	In this text box, specify the location of the files that are outside the folder with tests and are referenced in your tests through relative paths. Type the path manually or click the Browse button … and select the desired folder in the dialog that opens . This setting does not block the test execution because the location of tests is always specified through a full path to the corresponding files and/or directories.

Test Groups Tab

In this tab, appoint the groups of tests to execute when the tests are run according to the current run configuration.

Grouping tests is helpful, for example, to distinguish between tests to run in a production environment from those to run in your development environment. You just need to create two groups and then include or exclude them depending on the current environment.

To attach a test to a group, tag it with the @group annotation in the format:

```
/**
@group <group specification>
*/
```

To enable filtering tests based on their authors, tag the tests with the @author annotation.

Item	Description
Include/Exclude groups	Select this check box to enable configuring execution of test groups .
Group Name	In this read-only list, select the test group to be involved or skipped in testing.
Include	Select this check box to have the tests from the selected test group executed.
Exclude	Select this check box to have the tests from the selected test group skipped.

Remote Tab

In this tab, configure deployment of tests to a remote server.

Item	Description
Debug server	In this field, specify the configuration through which the debugging engine will interact with the server through HTTP. Actually, a configuration consist of a URL address of the target environment, the debugger type, and possibly mappings between folders in your project and their correspondences on the server. Choose a configuration from the drop-down list or click the Browse button … and define a new configuration in the Servers dialog box that opens.
Upload test directory before run	– When this check box is selected, IntelliJ IDEA automatically uploads tests to the default server according to the deployment options. – When the check box is cleared, you have to upload tests manually, see Uploading and Downloading Files . The check box is by default selected.
Remove test files after run	– When this check box is selected, IntelliJ IDEA automatically removes the executed tests from the server upon execution. – When the check box is cleared, tests remain on the server after execution and you will have to download them manually, see Uploading and Downloading Files .
Show transfer logs	Select this check box to have test deployment logged.

Toolbar

	Item	Shortcut	Description
+		Alt+Insert	Click this button to add a new configuration to the list.
–		Alt+Delete	Click this button to remove the selected configuration from the list.
		Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults		Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓		Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder		Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations		Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

	Item	Keyboard shortcut	Description
+		Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the

		`package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files .
	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: App Engine for PHP

In this dialog box, create configurations to run and debug PHP applications locally on the PHP Development Server before deploying them to the Google PHP Runtime Environment.

The dialog box is available only if the Google App Engine Support for PHP plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins.

- App Engine for PHP Specific Configuration Settings
- Command Line
- Toolbar
- Common options

App Engine for PHP Specific Configuration Settings

Item	Description
Name	In this text box, specify the name of the run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Host	In this field, specify the host to run the development server and the application on, the default is `localhost`.
Port	In this field, specify the port through which IntelliJ IDEA will communicate with the development server, the default port is `8080`.

Command Line

In this area, specify the settings for running and debugging your application on the PHP development server in the command-line mode.

Item	Description
Interpreter options	In this text box, specify the options to be passed to the PHP executable file. They override the default behavior of the PHP interpreter and/or ensure that additional activities are performed. If necessary, click and type the desired options in the Command Line Options dialog box. Type each option on a new line. When you close the dialog box, they are all displayed in the Command line options text box with spaces as separators.
Interpreter options	In this field, specify the options to be passed to the PHP executable file of the built-in PHP interpreter, see Command-Line Arguments for details.
Custom working directory	In this text box, specify the location of the files that are outside the folder with your sources and are referenced through relative paths. Type the path manually or click the Browse button and select the desired folder in the dialog that opens.
Environment variables	In this field, specify the environment variables be passed to the built-in server. See Environment Variables in Apache for details.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
−	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder. If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Plugin

Use this dialog to create a run/debug configuration to be used for running and debugging the plugin projects.

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Item	Description
VM options	Specify the string passed to the VM for launching the plugin. Usually this string contains the options such as `-mx` , `-verbose` , etc. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Program arguments	Type the list of arguments to be passed to the program, same way as if you were entering these parameters in the command line. Use the same rules as for specifying the VM options .
Use classpath of module	Select the module whose classpath should be used to run the application.
Use alternative JRE	Select this check box to enable defining another JRE than the JRE used by the current project / module.
Show idea.log	If selected, makes IntelliJ IDEA show content of the `idea.log` file in the console while running the plugin.

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
–	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	`Alt+Up` or `Alt+Down`	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts .

- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Play2 App

Use this dialog to create or edit run/debug configuration for Play 2 framework.

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Item	Description
Open in browser	Use this area to specify the following browser options: – Open in browser after compilation - this check box lets you open your compilation results in a browser. By default, the check box is selected. – Url To Open - this field shows you the default url address.
JVM Options	Use this field to specify JVM options. By default, the JVM options are specified in this field.
Environment variables	Use this field to set the environment variables.
Use non-default Play 2 install dir	Select this check box to use non-default Play 2 installation directory.
Debug Port	Use this field to specify the debug port. Play 2 framework uses 9999 as a default debug port.
Enable auto-reload	Select this check box to specify the auto-reload. In this case Play 2 framework detects changes in source files and recompiles them.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more

information, see Ant .

- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Remote

Use this dialog to create a run/debug configuration to be used for remote running or debugging processes (e.g. applications, servlets, plugin applets). Remote running/debugging enables you to connect to a running JVM.

The dialog box consists of the following tabs:

- Configuration tab
- Logs tab

Click here for the description of the options that are common for all run/debug configurations.

Configuration tab

Item	Description
Command line arguments for running remote JVM	These read-only fields show the command-line arguments to be used for running a remote JVM. IntelliJ IDEA suggests two arguments, depending on the JDK version used.
Transport: Socket	If the Socket transport is selected, specify host where the remote process is running and to which the debugger will connect. Specify also the port on remote host to which the debugger should connect.
Transport: Shared Memory	Having selected Shared memory transport, specify the shared memory address in the text field. This kind of transport is available for Windows only.
Debugger mode: Attach	Select this radio button is you want the debugger to connect to a running remote JVM.
Debugger mode: Listen	Select this radio button is you want the debugger to run as a server for remote JVM's which will connect to it to perform debugging. In this case the Host field will be disabled (if the Socket transport is selected). The Port value is considered as a port at which the debugger-"server" will listen to connections from the debugger-"clients".

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Resin

Run | Edit Configurations | + | Resin | Local or Remote

Resin run/debug configurations let you deploy and debug your applications on Resin . (The Resin Integration plugin must be enabled.)

- Name field and Share option
- Server tab for a local configuration
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .

JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Resin configuration template	Specify the location of the Resin configuration template. You can click … and select the necessary file in the dialog that opens .
Additional Resin command line	If necessary, specify additional commands for the server. If you need more room to type, click next to the field to open the Additional Resin command line dialog where the text entry area is larger.
Do not alter Resin configuration	Select this check box to disallow editing the Resin configuration file.
Charset	Specify the character set to be used.
JMX port	Specify the JMX server port.
Deploy mode	Select the deploy-mode (startup-mode) for applications on the server.
HTTP port	The server HTTP port. You may want to change the default port `80` (e.g. to `8080`).

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click … (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Charset	Specify the character set to be used.
JMX port	Specify the JMX server port.
Ping	Click this button to ping the JMX port on the server.
Remote staging	This section contains the settings related to staging . An example of remote staging settings for a mounted folder is provided after this table.
Type	Select the way the staging environment or host is accessed for transferring the application artifact or artifacts from your local computer. (In the user interface of IntelliJ IDEA this setting is also referred to as the connection type.) The available options are: – Same file system. Select this option if the target server is installed on your local computer. The artifacts in this case are deployed locally and, thus, don't need to be transferred to a remote host. – ftp. The File Transfer Protocol or Secure FTP is used. – Local or mounted folder. The staging environment is a local folder or is accessed as a mounted folder . If the list is empty, you have to enable the Remote Hosts Access plugin which supports the corresponding functionality.
Host	If Same file system is selected for Type, the only available option for Host is also Same file system. In all other cases, the list contains the existing configurations of the selected type . So each configuration corresponds to an individual (S)FTP connection, or a local or mounted folder. Select an existing configuration or create a new one. To create a new configuration: 1. Click … to the right of the list. 2. In the Deployment dialog , click +. 3. In the Add Server dialog, specify the configuration name, select the type, and click OK. 4. On the Connection tab , specify the settings in the Upload/download project files section. The rest of the settings don't matter. 5. Click OK in the Deployment dialog.
Path from root	The path to the staging folder relative to the local or mounted folder, or the root of the (S)FTP host. You can use … to select the folder in the Choose target path dialog. Note that if Same file system is selected for Type and Host, this setting doesn't need to be specified.
Remote connection	The settings for accessing deployed applications.

settings	
Host	The fully qualified domain name or the IP address of the Resin host.
Port	The server HTTP port.

An example of remote staging settings for a mounted folder

Assuming that:

- `C:\shared` is a shared folder on the remote host which is mounted to the local computer as the drive `X:`.
- The folder that you are going to use for staging is `C:\shared\staging`.

Here are the corresponding remote staging settings:

- Type: Local or mounted folder.
- Host: The configuration should be selected in which the value in the Folder field is `X:\` (the Upload/download project files section on the Connection tab of the Deployment dialog).
- Path from root: `staging`

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. To edit the settings for an artifact or external resource, select the corresponding item in the list and use the controls in the right-hand part of the tab. For more information, see the table below.

Item	Description
+(Alt+Insert)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens.
–(Alt+Delete)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(F4)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)
Deployment method	For local configurations: select the deployment method for the selected artifact or external resource (JMX or resin.xml).
Resin host name	For local configurations: specify the name or the IP address of the Resin host (for example, `localhost` or `127.0.0.1`).
Use default context name	For local configurations: select this check box if you want to use the default context root for the selected artifact or external resource. Otherwise, clear this check box and specify the context root in the Application context name field. Note that if the deployment method is JMX, the default context root is always used.
Application context name	For local configurations: specify the context root for the selected artifact or external resource.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window.

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window.
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens.
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog.
–	Click this button to remove the selected log entry from the list.
...	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use or to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click .
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
	Alt+Insert	Create a run/debug configuration.
	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.

You can group run/debug configurations by placing them into folders.
To create a folder, select the configurations to be grouped and click . Specify the name of the folder.

Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder.

To remove grouping, select a folder and click .

See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: SBT Task

IntelliJ IDEA lets you specify a run/debug configuration for SBT tasks.

Item	Description
Tasks	
VM parameters	If necessary, specify the string to be passed to the VM. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. If there is not enough space, you can click and enter the string in the dialog that opens.
Environment variables	Click to open the Environment Variables dialog box where you can create variables and specify their values.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder. If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant.

- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Scala

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

Configuration tab

Item	Description
Main class	In this text box, specify the fully qualified name of the class to be executed (passed to the JRE). Type the class name manually or click the Browse button to open the Choose Main Class dialog box, where you can find the desired class by name or search through the project.
VM options	In this text box, specify the string to be passed to the VM for launching an application. Usually this string contains the options such as `-mx` , `-verbose` , etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . The `-classpath` option specified in this field overrides the classpath of the module.
Program arguments	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options .
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens . Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. Note The list of environment variables may vary depending on the enabled plugins.
Environment variables	Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values. Note that you can copy-paste the contents of the Environment variables field without having to open the Environment Variables dialog box.
Use classpath of module	Select the module whose classpath should be used to run the application.
JRE	By default, the newest JDK from the module dependencies is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Enable capturing form snapshots	Select this check box to enable the GUI Designer to take snapshots of the GUI components , that can be afterwards converted into a form.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
↑ or ↓	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the ↑↓ buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration

irrespective of the compilation result.

- Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Scala Console

This section provides descriptions of the configuration-specific items for Scala console, as well as options that are common for all run/debug configurations.

Scala Console Options

Configuration tab

Item	Description
VM options	In this text box, specify the string to be passed to the VM for launching an application. Usually this string contains the options such as `-mx`, `-verbose`, etc. If necessary, click and type the desired string in the VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. The `-classpath` option specified in this field overrides the classpath of the module.
Console arguments	In this text box, type a list of arguments to be passed to the program in the format you would use in the command line. If necessary, click the button and type the desired arguments in the Program Parameters dialog box. Use the same rules as for specifying the VM options.
Working directory	In this text box, specify the current directory to be used by the running test. This directory is the starting point for all relative input and output paths. By default, the field contains the directory where the project file resides. To specify another directory, click the Browse button select the directory in the dialog that opens. Click this icon to view the list of available environment variables that you can use as a path to your working directory. In this case you do not need to specify any additional environment variables in the Environment variables field. Note The list of environment variables may vary depending on the enabled plugins.
Use classpath and SDK of module	Select the module whose classpath and SDK should be used to run the application.

Toolbar

Item	Shortcut	Description
	Alt+Insert	Click this button to add a new configuration to the list.
	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder. If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Scala Script

IntelliJ IDEA lets you specify a run/debug configuration for Scala scripts.

Script file	
Program parameters	
VM options	If necessary, specify the string to be passed to the VM. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`. If there is not enough space, you can click and enter the string in the dialog that opens.
Script Runner Args (e.g. -deprecation)	
Working directory	Specify the directory that will act as the current directory when running the test. It will act as the root directory for all relative input and output paths. By default, the directory where the project file resides, is used as a working directory. Type directory name, or click and select the desired directory in the dialog that opens.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder. If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.

		See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Scala Test

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

Configuration tab

Item	Description
Test kind	From this drop-down list, select the scope for your tests and fill in the fields depending on your selection. You can choose from the following options: – Test name - select this option to run the specified Scala test. The name of the test appears in the Test Name field. – All in package - select this option to run all Scala tests in the specified package. Fill in the following fields: – Test Package - specify the name of the package to be tested. – Search for tests - use this drop-down list to select the scope of your search. – Class - select this option to run all tests in a class. Specify the fully qualified name of the class to be launched in the Test Class field. Type the class name or click … and select the desired class in the dialog that opens.
VM parameters	If necessary, specify the string to be passed to the VM. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . If there is not enough space, you can click and enter the string in the dialog that opens.
Environment variables	Click … to open the Environment Variables dialog box where you can create variables and specify their values.
Test options	Use this field to specify the additional test options. If there is not enough space, you can click and enter the string in the dialog that opens.
Working Directory	Specify the directory that will act as the current directory when running the test. It will act as the root directory for all relative input and output paths. By default, the directory where the project file resides, is used as a working directory. Type directory name, or click … and select the desired directory in the dialog that opens.
Use classpath and SDK of module	From this drop-down list, select the module whose classpath will be used to run the application.
Print information messages to console	Select this check box if you want to print information messages to the Scala console.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool

window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise.

		If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Specs2

IntelliJ IDEA lets you specify a Run/Debug configuration for Specs2 .

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

Configuration tab

Item	Description
Test kind	From this drop-down list, select the scope for your tests and fill in the fields depending on your selection. You can choose from the following options: – Test name - select this option to run the specified Scala test. The name of the test appears in the Test Name field. – All in package - select this option to run all Scala tests in the specified package. Fill in the following fields: – Test Package - specify the name of the package to be tested. – Search for tests - use this drop-down list to select the scope of your search. – Class - select this option to run all tests in a class. Specify the fully qualified name of the class to be launched in the Test Class field. Type the class name or click … and select the desired class in the dialog that opens.
VM parameters	If necessary, specify the string to be passed to the VM. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . If there is not enough space, you can click and enter the string in the dialog that opens.
Environment variables	Click … to open the Environment Variables dialog box where you can create variables and specify their values.
Test options	Use this field to specify the additional test options. If there is not enough space, you can click and enter the string in the dialog that opens.
Working Directory	Specify the directory that will act as the current directory when running the test. It will act as the root directory for all relative input and output paths. By default, the directory where the project file resides, is used as a working directory. Type directory name, or click … and select the desired directory in the dialog that opens.
Use classpath and SDK of module	From this drop-down list, select the module whose classpath will be used to run the application.
Print information messages to console	Select this check box if you want to print information messages to the Scala console.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a

particular module is specified in the run/debug configuration, and the Make Project command otherwise.
If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration.

- Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result.
- Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Spring Boot

Use this dialog box to create configurations used to run or debug Spring Boot applications.

This dialog box consists of the following areas:

- Configuration tab
- Logs tab
- Before launch options

Configuration tab

Use this tab to configure general run/debug configuration settings, and the specific Spring Boot settings.

Item	Description
Main class	In this text box, specify the fully qualified name of the class to be executed (passed to the JRE). Enter the class name manually or click the Browse button and search for the desired class by name or through the project.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Program arguments	Enter a list of arguments to be passed to the program in the format you would use in the command line. If there is not enough space, click the button and enter the parameters in the Program Parameters dialog box. Use the same rules as for specifying the VM options .
Working directory	If necessary, specify a directory to be used by the running text. This directory is the starting point for all relative input and output paths. By default, this is the project directory. To specify a different directory, click the Browse button and select a directory in the dialog that opens .
Environment variables	Click the Browse button to open the Environment Variables dialog where you can create variables and specify their values.
Use classpath of module	Select the module whose classpath will be used to run the application.
JRE	By default, the newest JDK from the module dependencies is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
Enable debug output	Select this option to enable logging of the debug output.
Hide Banner	Select this option to hide the startup banner printed by Spring Boot when starting a run/debug session.
Active Profiles	If necessary, specify active Spring profiles that you want to use in this run/debug configuration.
Override parameters	Use this table to specify which Spring Boot configuration parameters you want to override. Such parameters are normally defined in configuration files. Listing them in a run/debug configuration allows you to easily switch parameters by modifying the run/debug configuration instead of config files, or have multiple run/debug configurations with different parameter values. When you add a new parameter and start typing its name, IntelliJ IDEA provides content completion and displays a list of parameters that match the string you've entered. Use the following controls: – : use the checkboxes in the Enabled column to select which parameters you want to override. You can simply deselect the parameters you do not need to override at the moment without removing them and losing their values. – : click this icon to add a parameter you want to override to the list. – : click this icon to remove a parameter from the list. – : click these icons to move the selected parameter up or down in the list.

Logs tab

Use this tab to specify which log files generated while running or debugging must be displayed in the console, that is in the dedicated tabs of the Run tool window and the Debug tool window .

Item	Description
Is active	Select checkboxes in this column to have the corresponding log entries displayed in the Run tool window and the Debug tool window .
Log File Entry	The read-only fields in this column list the log files to be shown. This list can contain: – Full paths to specific files – Ant patterns that define the range of files to be displayed – Aliases that substitute full paths or patterns. These aliases are also displayed in the tab headers where the corresponding log files are shown. Note that if a log entry pattern defines more than one file, the tab header shows the name of the file instead of the lof entry alias.
Skip Content	Select the checkboxes in this column to skip the previous contents of the selected log.
Save console output to file	Select this option to save the console output to a specified location. Type the path manually, or click the Browse button and select a location in the dialog that opens .
Show console when standard out changes	Select this option to activate the output console and bring it forward if the associated process is writing to Standard.out.
Show console when standard error changes	Select this option to active the output console and bring it forward if the associated process is writing to Standard.err.
+	Click this button to add a new log file entry. Specify the associated properties in the Edit Log Files Aliases dialog .
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the selected log file entry. This button only appears when an entry is selected.

Before launch options

Use this section to specify which tasks must be carried out before starting the run/debug session. The tasks you specify will be performed in the order that they appear in the list.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select from the following task types: – Run External tool: select this option to run an application that is external to IntelliJ IDEA. Select the application(s) you want to run in the dialog that opens. If the required application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make: select this option to compile the project or module. If a particular module is specified in the run/debug configuration, the Make Module command will be executed. Otherwise, the Make Project command will be performed. – Make, no error check: the same as the Make option, but IntelliJ IDEA will try to launch the run/debug session irrespective of the compilation result. – Build Artifacts: select this option to build one or several artifacts. Select the artifacts you want to build in the dialog that opens. For more information, see Working with Artifacts . – Run Another Configuration: select this option to have a different run/debug configuration executed first. Select the configuration that you want to run in the dialog that opens. This option is available only if you have already at least one run/debug configuration in the current project. – Make Project: select this option to compile all files that have been modified since the last compilation, and dependent source files. – Run Ant target: select this option to run an Ant target. Select the target you want to run in the dialog that opens. For more information, see Ant . – Run Gradle task: select this option to run a Gradle task. In the dialog that opens, specify the Gradle project, the tasks you want to run, and, if necessary the VM options and script parameters. For more information, see Gradle . – Generate CoffeeScript Source Maps: select this option to generate the source maps for your CoffeeScript sources. Specify where your CoffeeScript source files are located in the dialog that opens. For more information, see CoffeeScript Support . – Run Maven Goal: select this option to run a Maven goal. Select the goal you want to run in the dialog that opens. For more information, see Maven . – Run Remote External tool: select this option to run a remote application external to IntelliJ IDEA. Select the remote application(s) you want to run in the dialog that opens. If the required application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
↑/↓	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list.
Show this page	N/A	Select this check box to show the run/debug settings before starting a run/debug session.

Run/Debug Configuration: Spring DM Server

In this part:

- Run/Debug Configuration: Spring DM Server (Local)
- Run/Debug Configuration: Spring DM Server (Remote)

Run/Debug Configuration: Spring DM Server (Local)

Run | Edit Configurations | + | Spring dmServer | Local

Spring dmServer run/debug configurations let you deploy and debug your applications on SpringSource dm Server and Virgo. (The dmServer Support plugin must be enabled.)

Note that if you are starting the server by means of the default script `startup.bat` or `startup.sh`, you should properly set the environment variable `JAVA_HOME`.

– Name field and Share option
– Server tab for a local configuration
– Deployment tab
– Logs tab
– Code Coverage tab
– Startup/Connection tab for a local configuration
– Before Launch options
– Toolbar

See also, Working with Server Run/Debug Configurations.

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.) If the message *Error: JMX arguments are incompatible with IDEA* is shown in the lower part of the dialog, click Fix. As a result, the necessary changes are made to the script `bin/dmk.bat` or `dmk.sh`.
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`.
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.

On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Port	Specify the HTTP server port.
JMX Port	Specify the JMX server port.
JMX User	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the JMX server port.
JMX Password	Specify the password of the JMX user .
Local staging / Repository target	Select the watched repository to place your plans (artifacts) to.

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. To edit the settings for an artifact or external resource, select the corresponding item in the list and use the controls in the right-hand part of the tab. For more information, see the table below.

Item	Description
+ (Alt+Insert)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
– (Alt+Delete)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(F4)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
...	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use + or + to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click –.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a local configuration

Item	Description
Run / Debug / Coverage	Use to switch between the settings for the run, debug and code coverage modes.
Startup script	Specify the script to be used to start the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Startup Script dialog which shows the contents of the Startup script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Specify the script to be used to stop the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Shutdown Script dialog which shows the contents of the Shutdown script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Pass environment variables	To pass specific variables to the server environment, select this check box and specify the variables: – To add a variable, click + and specify the variable name and value in the Name and Value fields respectively. – To remove a variable from the list, select the variable and click –.
Port	Use this field to change the debugger port.
Debugger Settings	Click this button to edit the debugger options on the Debugger page of the Settings dialog .

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
−	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click −. See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: Spring DM Server (Remote)

Run | Edit Configurations | + | Spring dmServer | Local or Remote

Spring dmServer run/debug configurations let you deploy and debug your applications on SpringSource dm Server and Virgo . (The dmServer Support plugin must be enabled.)

Note that if you are starting the server by means of the default script `startup.bat` or `startup.sh` , you should properly set the environment variable `JAVA_HOME` .

- Name field and Share option
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
JMX Port	Specify the JMX server port.
Ping	Click this button to ping the JMX port on the server.
JMX User	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the JMX server port.
JMX Password	Specify the password of the JMX user .
Remote staging	This section contains the settings related to staging . An example of remote staging settings for a mounted folder is provided after this table.
Type	Select the way the staging environment or host is accessed for transferring the application artifact or artifacts from your local computer. (In the user interface of IntelliJ IDEA this setting is also referred to as the connection type.) The available options are: – Same file system. Select this option if the target server is installed on your local computer. The artifacts in this case are deployed locally and, thus, don't need to be transferred to a remote host. – ftp. The File Transfer Protocol or Secure FTP is used. – Local or mounted folder. The staging environment is a local folder or is accessed as a mounted folder . If the list is empty, you have to enable the Remote Hosts Access plugin which supports the corresponding functionality.

Host	If Same file system is selected for Type, the only available option for Host is also Same file system. In all other cases, the list contains the existing configurations of the selected type . So each configuration corresponds to an individual (S)FTP connection, or a local or mounted folder. Select an existing configuration or create a new one. To create a new configuration: 1. Click ... to the right of the list. 2. In the Deployment dialog , click +. 3. In the Add Server dialog, specify the configuration name, select the type, and click OK. 4. On the Connection tab , specify the settings in the Upload/download project files section. The rest of the settings don't matter. 5. Click OK in the Deployment dialog.
Deployment access	When deploying to the remote host, the application artifact or artifacts are placed into a staging folder (deployment target folder). This folder should be accessible to the server. The settings in this section define the location of this staging folder.
Path from root	The path to the staging folder relative to the local or mounted folder, or the root of the (S)FTP host. You can use ... to select the folder in the Choose target path dialog.
Mapped as	The absolute path to the staging folder in the local file system of the remote host. The path should be specified as a URL, e.g. `file:///C:/shared/staging`
Repository access	When deploying to the remote host, the plans (artifacts) are placed into a watched repository. The settings in this section describe this repository.
Path from root	The path to the repository folder relative to the local or mounted folder, or the root of the (S)FTP host. You can use ... to select the folder in the Choose target path dialog.
Name	Specify the name of the watched repository the way it is defined in the configuration of the remote server.
Remote connection settings	The settings for accessing deployed applications.
Host	The fully qualified domain name or the IP address of the server host.
Port	The server HTTP port.

An example of remote staging settings for a mounted folder

Assuming that:

- `C:\shared` is a shared folder on the remote host which is mounted to the local computer as the drive `X:` .
- The folder that you are going to use for staging is `C:\shared\staging` .

Here are the corresponding remote staging settings:

- Type: Local or mounted folder.
- Host: The configuration should be selected in which the value in the Folder field is `X:\` (the Upload/download project files section on the Connection tab of the Deployment dialog).
- Staging/Path from root: `staging`
- Staging/Mapped as: `file:///C:/shared/staging`

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. To edit the settings for an artifact or external resource, select the corresponding item in the list and use the controls in the right-hand part of the tab. For more information, see the table below.

Item	Description
+ (`Alt+Insert`)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
– (`Alt+Delete`)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(`F4`)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.

Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
−	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
−	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
−	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.

You can group run/debug configurations by placing them into folders.
To create a folder, select the configurations to be grouped and click . Specify the name of the folder.

Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder.

To remove grouping, select a folder and click .

See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: Spy-js

In this dialog box, create configurations for tracing Web applications using the Spy-js tool.

On this page:

- Getting Access to the Run/Debug Configuration: Spy-js Dialog Box
- Spy-js-Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: Spy-js Dialog Box

1. Download and install Node.js because it is used by the Spy-js trace server.
2. Install and enable the Spy-js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Spy-js-Specific Configuration Settings

Item	Description
Node interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file. With Spy-js, it is recommended to use Node.js version 0.10.24 or higher.
Trace server port	In this spin box, specify the port on which Spy-js will act as a proxy server. This port number must be the same as your system proxy port. If the Automatically configure system proxy check box is selected, the specified port number is automatically set for the system proxy server. Otherwise you will have to specify the value of the field in the system proxy settings manually. The trace server port is filled in automatically. To avoid port conflicts, it is recommended that you accept the suggested value and keep the Automatically configure system proxy check box selected.
Use	In this drop-down list, specify the way to configure a tracing session. – To have Spy-js apply its internal predefined configuration, choose Default configuration. – To have your custom manually created configuration applied, choose the Configuration file option and then specify the location of your custom configuration file in the Configuration field below. A configuration file is a JavaScript file with the extension `.js` or `.conf.js` that contains valid JavaScript code that meets the Spy-js configuration requirements . If IntelliJ IDEA detects files with the extension `.conf.js` in the project, these files are displayed in the drop-down list. Type the path to the configuration file manually or click the Browse button ... and choose the location in the dialog box that opens. Once specified, a configuration file is added to the drop-down list so you can get if next time from the list instead of specifying the path.
URL to trace	In this field, specify the URL address of the Web page to capture events on. By default, the field is empty. This means that Spy-js captures events on all the currently opened Web pages. If you want to restrict the tracing to a certain page, specify its URL address. Type the address manually or choose it from the drop-down list, if it has been once specified in the configuration.
Automatically configure system proxy	– When this check box is selected, the system proxy server is activated automatically with the port specified in the Trace server port field. – Clear this check box to specify proxy settings manually. See how to configure proxy settings manually on Windows , Mac , Ubuntu , iOS , Android , Windows Phone . Please note that some desktop browsers have their own screens for proxy settings configuration. The check box is selected by default, and it is strongly recommended that you accept this setting and have the proxy configured automatically.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the

run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise.

If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Spy-js for Node.js

In this dialog box, create configurations for tracing Node.js applications using the Spy-js tool.

On this page:

- Getting Access to the Run/Debug Configuration: Spy-js for Node.js Dialog Box
- Spy-js for Node.js- Specific Configuration Settings
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: Spy-js for Node.js Dialog Box

1. Download and install Node.js because it is used by the Spy-js trace server.
2. Install and enable the Spy-js plugin. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Spy-js for Node.js- Specific Configuration Settings

Item	Description
Node interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file. With Spy-js, it is recommended to use Node.js version 0.10.24 or higher. To enable tracing ECMAScript 6 scripts, use Node.js version 0.11.13 or higher and specify `--harmony` as a Node parameter.
Node parameters	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. For example, to enable tracing ECMAScript 6 scripts, specify `--harmony` as a Node parameter. Note that Node.js must be version 0.11.13 or higher.
Working directory	In this field, specify the working directory of the application. All references in the starting NodeJS application file, for example, includes, will be resolved relative to this folder, unless such references use full paths. By default, the field shows the project root folder. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button ... and select the location in the dialog box, that opens.
JavaScript file	In this field, specify the full path to the file to start running the application from. If you are going to trace CoffeeScript, specify the path to the generated JavaScript file. The file can be generated externally or through transpilation using file watchers. For more details, see Transpiling CoffeeScript to JavaScript .
Application parameters	In this text box, type the NodeJS-specific arguments to be passed to the application start file through the process.argv array.
Environment variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button ... to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: – To define a new variable, click the Add toolbar button + and specify the variable name and value. – To discard a variable definition, select it in the list and click the Delete toolbar button —. – Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: – `NODE_PATH` : A `:` -separated list of directories prefixed to the module search path. – `NODE_MODULE_CONTEXTS` : Set to 1 to load modules in their own global contexts. – `NODE_DISABLE_COLORS` : Set to 1 to disable colors in the REPL.
Configuration file	From this drop-down list, choose the file with the configuration settings to apply to the tracing session. A configuration file is a JavaScript file with the extension `.js` or `.conf.js` that contains valid JavaScript code that meets the Spy-js configuration requirements . If IntelliJ IDEA detects files with the extension `.conf.js` in the project, these files are displayed in the drop-down list. Type the path to the configuration file manually or click the Browse button ... and choose the location in the dialog box that opens. Once specified, a configuration file is added to the drop-down list so you can get if next time from the list instead of specifying the path.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
—	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: TestNG

The TestNG run/debug configuration is used to launch the tests that comply with the TestNG framework. The dialog consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

Configuration tab

The composition of this tab depends on the selected testing scope - package, project, etc.

Item	Description
All in package	Run all tests in a package. Package. The fully qualified name of the package. In whole project. IntelliJ IDEA will look for the tests in all the modules. In single module. IntelliJ IDEA will look for the tests only in the module that is selected in the Use classpath of module field . Across module dependencies. The same as the previous option plus the modules that depend on that module.
Suite	Run a test suite. Suite. Specify the corresponding `testng.xml` file.
Group	Run a test group. Group. The group to be run. Learn more about TestNG groups .
Class	Run a test class. Class. The fully qualified name of the test class to be run.
Method	Run a test method. Class. The fully qualified name of the test class. Method. The name of the method to be run.
Pattern	Run the tests that conform to the specified pattern. Pattern. Form the pattern by clicking + and then selecting one or more TestNG test classes. Alternatively, click and type the pattern in the dialog that opens.
Output directory	The directory in which test reports will be generated.
JDK Settings	
VM options	Options and arguments to be passed to the JVM in which the tests run. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . The `-classpath` option specified in this field overrides the classpath of the module.
Test runner parameters	Arguments to be passed to the test runner. Use the same rules as for specifying the VM options .
Working directory	The current working directory for the tests.
Environment variables	The environment variables to be passed to the corresponding JVM.
Use classpath of module	The module whose classpath is used when running the tests.
JRE	The JRE to be used.
Parameters	
Properties file	Specify the `.properties` file to be passed to TestNG.
Name - Value	Additional paramenters as key - value pairs.
Listeners	
+ −	Use these icons to make up a list of listeners.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.

Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.

This check box is not available when editing the run/debug configuration defaults.

Single instance only

If this check box is selected, this run/debug configuration cannot be launched more than once.
Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch

Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: TestNG Test Discovery

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

It also contains a set of options that are common for all run/debug configurations and a toolbar that lets you manage your configurations.

Tip You can use `Ctrl+Space` to let IntelliJ IDEA help you fill in the fields in this dialog. (In the editor, `Ctrl+Space` is used for code completion .)

Configuration tab

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: - Full paths to specific files. - Ant patterns that define the range of files to be displayed. - Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
−	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	`Alt+Insert`	Click this button to add a new configuration to the list.
−	`Alt+Delete`	Click this button to remove the selected configuration from the list.
	`Ctrl+D`	Click this button to create a copy of the selected configuration.

	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.

Alt+Up	Click this icon to move the selected task one line up in the list.
Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Tomcat Server

Run | Edit Configurations | + | Tomcat Server | Local or Remote

Tomcat Server run/debug configurations let you deploy and debug your applications on Apache Tomcat . (The Tomcat and TomEE Integration plugin must be enabled.)

- Name field and Share option
- Server tab for a local configuration
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a local configuration
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .

JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
HTTP port	The server HTTP port.
HTTPs port	The server HTTPS port.
JMX port	The server JMX port.
AJP port	The server AJP port.
Deploy applications configured in Tomcat instance	The Tomcat configuration files, among other things, may list the applications that should be deployed at the server start. If this check box is selected, all the applications so listed will be deployed on the server in addition to the artifacts specified on the Deployment tab . If the check box is not selected, only the artifacts and the external resources specified on the Deployment tab will be deployed.
Preserve sessions across restarts	Select this check box to preserve active HTTP sessions when restarting the server.

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
JMX Port	The server JMX port. If you are not deploying anything with this run configuration, you don't need to specify this port.
Remote staging	This section contains the settings related to staging . An example of remote staging settings for a mounted folder is provided after this table.
Type	Select the way the staging environment or host is accessed for transferring the application artifact or artifacts from your local computer. (In the user interface of IntelliJ IDEA this setting is also referred to as the connection type.) The available options are: – Same file system. Select this option if the target server is installed on your local computer. The artifacts in this case are deployed locally and, thus, don't need to be transferred to a remote host. – ftp. The File Transfer Protocol or Secure FTP is used. – Local or mounted folder. The staging environment is a local folder or is accessed as a mounted folder . If the list is empty, you have to enable the Remote Hosts Access plugin which supports the corresponding functionality.
Host	If Same file system is selected for Type, the only available option for Host is also Same file system. In all other cases, the list contains the existing configurations of the selected type . So each configuration corresponds to an individual (S)FTP connection, or a local or mounted folder. Select an existing configuration or create a new one. To create a new configuration: 1. Click ... to the right of the list. 2. In the Deployment dialog , click +. 3. In the Add Server dialog, specify the configuration name, select the type, and click OK. 4. On the Connection tab , specify the settings in the Upload/download project files section. The rest of the settings don't matter. 5. Click OK in the Deployment dialog.
Staging	When deploying to the remote host, the application artifact or artifacts are placed into a staging folder which should be accessible to Tomcat. The settings in this section define the location of this staging folder. Note that if Same file system is selected for Type and Host, no settings in this section need to be specified.
Path from root	The path to the staging folder relative to the local or mounted folder, or the root of the (S)FTP host. You can use ... to select the folder in the Choose target path dialog.
Mapped as	The absolute path to the staging folder in the local file system of the remote host.
Remote connection	The settings for accessing deployed applications.

settings	
Host	The fully qualified domain name or the IP address of the Tomcat host.
Port	The server HTTP port.

An example of remote staging settings for a mounted folder

Assuming that:

- `C:\shared` is a shared folder on the remote host which is mounted to the local computer as the drive `X:`.
- The folder that you are going to use for staging is `C:\shared\staging`.

Here are the corresponding remote staging settings:

- Type: Local or mounted folder.
- Host: The configuration should be selected in which the value in the Folder field is `X:\` (the Upload/download project files section on the Connection tab of the Deployment dialog).
- Staging/Path from root: `staging`
- Staging/Mapped as: `C:\shared\staging`

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. To edit the settings for an artifact or external resource, select the corresponding item in the list and use the controls in the right-hand part of the tab. For more information, see the table below.

Note that deployment to a remote server is supported only for Tomcat 5 or later versions. Also note that to be able to deploy applications to a remote Tomcat server, you should enable the JMX support on the server. To do that, you should pass the following VM options to the server Java process:

```
-Dcom.sun.management.jmxremote=
-Dcom.sun.management.jmxremote.port=1099
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
-Djava.rmi.server.hostname=<host>
```

where `<host>` is the server hostname (domain name) or IP address.

If `catalina.bat` or `catalina.sh` is used to start the server, these options may be passed to the server using the `CATALINA_OPTS` environment variable.

Item	Description
+ (`Alt+Insert`)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
– (`Alt+Delete`)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(`F4`)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)
Application context	Specify the context root for an artifact or external resource: select the artifact or the resource, and type or select the context root.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.

+

	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use or to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click .
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a local configuration

Item	Description
Run / Debug / Coverage	Use to switch between the settings for the run, debug and code coverage modes.
Startup script	Specify the script to be used to start the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Startup Script dialog which shows the contents of the Startup script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Specify the script to be used to stop the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Shutdown Script dialog which shows the contents of the Shutdown script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Pass environment variables	To pass specific variables to the server environment, select this check box and specify the variables: – To add a variable, click and specify the variable name and value in the Name and Value fields respectively. – To remove a variable from the list, select the variable and click .
Port	Use this field to change the debugger port.
Debugger Settings	Click this button to edit the debugger options on the Debugger page of the Settings dialog .

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.

Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
↑↓	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
–	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
↑↓	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use ↑ and ↓. You can also drag a configuration into a folder. To remove grouping, select a folder and click –. See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: TomEE Server

Run | Edit Configurations | + | TomEE Server | Local or Remote

TomEE Server run/debug configurations let you deploy and debug your applications on TomEE . (The Tomcat and TomEE Integration plugin must be enabled.)

- Name field and Share option
- Server tab for a local configuration
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a local configuration
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
JRE	By default, the project JDK is used to run the application. If you want to specify an alternative JDK or JRE here, select it from the drop-down list.
HTTP port	The server HTTP port.

HTTPs port	The server HTTPS port.
JMX port	The server JMX port.
AJP port	The server AJP port.
Deploy applications configured in Tomcat instance	The TomEE configuration files, among other things, may list the applications that should be deployed at the server start. If this check box is selected, all the applications so listed will be deployed on the server in addition to the artifacts specified on the Deployment tab . If the check box is not selected, only the artifacts and the external resources specified on the Deployment tab will be deployed.
Preserve sessions across restarts	Select this check box to preserve active HTTP sessions when restarting the server.

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
JMX Port	The server JMX port.
Remote staging	This section contains the settings related to staging . An example of remote staging settings for a mounted folder is provided after this table.
Type	Select the way the staging environment or host is accessed for transferring the application artifact or artifacts from your local computer. (In the user interface of IntelliJ IDEA this setting is also referred to as the connection type.) The available options are: – Same file system. Select this option if the target server is installed on your local computer. The artifacts in this case are deployed locally and, thus, don't need to be transferred to a remote host. – ftp. The File Transfer Protocol or Secure FTP is used. – Local or mounted folder. The staging environment is a local folder or is accessed as a mounted folder . If the list is empty, you have to enable the Remote Hosts Access plugin which supports the corresponding functionality.
Host	If Same file system is selected for Type, the only available option for Host is also Same file system. In all other cases, the list contains the existing configurations of the selected type . So each configuration corresponds to an individual (S)FTP connection, or a local or mounted folder. Select an existing configuration or create a new one. To create a new configuration: 1. Click ... to the right of the list. 2. In the Deployment dialog , click +. 3. In the Add Server dialog, specify the configuration name, select the type, and click OK. 4. On the Connection tab , specify the settings in the Upload/download project files section. The rest of the settings don't matter. 5. Click OK in the Deployment dialog.
Staging	When deploying to the remote host, the application artifact or artifacts are placed into a staging folder which should be accessible to TomEE. The settings in this section define the location of this staging folder. Note that if Same file system is selected for Type and Host, no settings in this section need to be specified.
Path from root	The path to the staging folder relative to the local or mounted folder, or the root of the (S)FTP host. You can use ... to select the folder in the Choose target path dialog.
Mapped as	The absolute path to the staging folder in the local file system of the remote host.
Remote connection settings	The settings for accessing deployed applications.
Host	The fully qualified domain name or the IP address of the TomEE host.
Port	The server HTTP port.

An example of remote staging settings for a mounted folder

Assuming that:

`C:\shared` is a shared folder on the remote host which is mounted to the local computer as the drive `Y:`

– `C:\shared` is a shared folder on the remote host which is mounted to the local computer as the drive `X:`.
– The folder that you are going to use for staging is `C:\shared\staging`.

Here are the corresponding remote staging settings:

– Type: Local or mounted folder.
– Host: The configuration should be selected in which the value in the Folder field is `X:\` (the Upload/download project files section on the Connection tab of the Deployment dialog).
– Staging/Path from root: `staging`
– Staging/Mapped as: `C:\shared\staging`

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. To edit the settings for an artifact or external resource, select the corresponding item in the list and use the controls in the right-hand part of the tab. For more information, see the table below.

Item	Description
+(`Alt+Insert`)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
–(`Alt+Delete`)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(`F4`)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)
Application context	Specify the context root for an artifact or external resource: select the artifact or the resource, and type or select the context root.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use or to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click –.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a local configuration

Item	Description
Run / Debug / Coverage	Use to switch between the settings for the run, debug and code coverage modes.
Startup script	Specify the script to be used to start the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Startup Script dialog which shows the contents of the Startup script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click … and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Specify the script to be used to stop the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Shutdown Script dialog which shows the contents of the Shutdown script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click … and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Pass environment variables	To pass specific variables to the server environment, select this check box and specify the variables: – To add a variable, click + and specify the variable name and value in the Name and Value fields respectively. – To remove a variable from the list, select the variable and click —.
Port	Use this field to change the debugger port.
Debugger Settings	Click this button to edit the debugger options on the Debugger page of the Settings dialog .

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .

	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
	Alt+Insert	Create a run/debug configuration.
	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click . See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: utest

IntelliJ IDEA lets you specify a Run/Debug configuration for utest .

This section provides descriptions of the configuration-specific items , as well as the toolbar and options that are common for all run/debug configurations.

The dialog box consists of the following tabs:

- Configuration tab
- Code Coverage tab
- Logs tab

Configuration tab

Item	Description
Test kind	From this drop-down list, select the scope for your tests and fill in the fields depending on your selection. You can choose from the following options: – Test name - select this option to run the specified Scala test. The name of the test appears in the Test Name field. – All in package - select this option to run all Scala tests in the specified package. Fill in the following fields: – Test Package - specify the name of the package to be tested. – Search for tests - use this drop-down list to select the scope of your search. – Class - select this option to run all tests in a class. Specify the fully qualified name of the class to be launched in the Test Class field. Type the class name or click … and select the desired class in the dialog that opens.
VM parameters	If necessary, specify the string to be passed to the VM. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . If there is not enough space, you can click and enter the string in the dialog that opens.
Environment variables	Click … to open the Environment Variables dialog box where you can create variables and specify their values.
Test options	Use this field to specify the additional test options. If there is not enough space, you can click and enter the string in the dialog that opens.
Working Directory	Specify the directory that will act as the current directory when running the test. It will act as the root directory for all relative input and output paths. By default, the directory where the project file resides, is used as a working directory. Type directory name, or click … and select the desired directory in the dialog that opens.
Use classpath and SDK of module	From this drop-down list, select the module whose classpath will be used to run the application.
Print information messages to console	Select this check box if you want to print information messages to the Scala console.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Item	Description
Choose code coverage runner	Select the desired code coverage runner. By default, IntelliJ IDEA uses its own coverage engine with the Sampling mode. You can also choose JaCoCo or Emma for calculating coverage.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line. If this check box is selected, becomes available on the toolbar of the coverage statistic pop-up window. Note This option is only available for the Tracing mode of code coverage measurement for the testing run/debug configurations. Refer to the section Viewing Code Coverage Results .
Merge data with previous results	When you run your unit testing or application configuration several times, use this item to calculate statistics in the Project View, taking into account the statistics of each time you have run the configuration. Tip Finally, the line is considered *covered* if it is covered at least once.
Packages and classes to record code coverage data	Click and buttons to specify classes and packages to be measured. You can also remove classes and packages from the list by selecting them in the list and clicking the button.
Enable coverage in test folders.	If this check box is selected, the folders marked as test are included in the code coverage analysis.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
–	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a

particular module is specified in the run/debug configuration, and the Make Project command otherwise.
If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration.

- Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result.
- Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built.
  See also, Working with Artifacts .
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
  This option is available only if you have already at least one run/debug configuration in the current project.
- Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant .
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
  Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
  Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
  - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
  - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support .
- Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run.
  For more information, see Maven .
- Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.

	`Alt+Delete`	Click this icon to remove the selected task from the list.
	`Enter`	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	`Alt+Up`	Click this icon to move the selected task one line up in the list.
	`Alt+Down`	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: WebLogic Server

Run | Edit Configurations | + | WebLogic Server | Local or Remote

WebLogic Server run/debug configurations let you deploy and debug your applications on Oracle WebLogic Server . (The WebLogic Integration plugin must be enabled.)

- Name field and Share option
- Server tab for a local configuration
- Server tab for a remote configuration
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a local configuration
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations .

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.)

	The options other than Do nothing have the same meanings as in the case of the Update action .
User	Specify the user name associated with the domain administrator's account.
Password	Specify the password of the domain administrator.

Domain Path and associated settings	Specify the path to the desired WebLogic domain directory.
	Also specify which of the server instances in your WebLogic domain should be started and provide the associated settings. Note that the server instance that is started is also the one to which your artifacts are deployed .
	
	The available options depend on the domain structure:
	– The domain contains only one WebLogic Server instance, the Administration Server:
	  – Admin Server. Select the name of the Administration Server from the list.
	– The domain contains the Administration Server and one or more Managed Servers:
	  – Server to launch. Select the type of the server instance that should be started and specify the associated settings:
	  – Admin. The Administration Server.
	  – Admin Server. Select the name of the Administration Server from the list.
	  – Managed. One of the Managed Servers.
	  – Managed server. Select the name of the desired Managed Server from the list.
	  – Admin server host. Specify the name or the IP address of the Administration Server host (e.g. `localhost` or `127.0.0.1`).
	  – Admin server port. Specify the Administration Server port.
	
	By the time you execute the run/debug configuration which starts a Managed Server, the Administration Server in the corresponding domain must be already up and running.

Server tab for a remote configuration

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
User	Specify the user name associated with the domain administrator's account.
Password	Specify the password of the domain administrator.

Connect to and associated settings	Specify the server instance or cluster the project artifacts should be deployed to.
	Select the target category of the server instances and specify the associated settings:
	– admin server. The Administration Server.
	  – Server. Specify the name of the Administration Server.
	– managed server. One of the Managed Servers.
	  – Managed Server name. Specify the name of the target Managed Server.
	  – Admin Server host. Specify the DNS name or the IP address of the Administration Server host.
	  – Admin Server port. Specify the Administration Server port.
	– cluster. One of the Managed Server clusters.
	  – Cluster name. Specify the name of the target cluster.
	  – Admin Server name. Specify the name of the Administration Server.
	
	For the run/debug configuration to be functional, the following server instances in the target domain must be up and running:
	– The Administration Server (in all cases).
	– The target Managed Server if you are deploying the artifacts to a Managed Server.
	– At least one of the Managed Servers in the target cluster if you are deploying the artifacts to a cluster.

Test Connection	Click this button to test the connection with the server to make sure that the current settings are correct.
Host	The fully qualified domain name or the IP address of the host to which the applications are deployed: – If you are deploying the applications to the Administration Server or a Managed Server cluster, this is the Administration Server host. – If you are deploying the applications to a Managed Server, this is the Managed Server host.
Port	The server HTTP port.

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. To edit the settings for an artifact or external resource, select the corresponding item in the list and use the controls in the right-hand part of the tab. For more information, see the table below.

Item	Description
+ (`Alt+Insert`)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
– (`Alt+Delete`)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(`F4`)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)
Deployment method	For local run configurations: – Weblogic.Deployer. The server admin interface is used. – Auto deploy. The artifact is copied to the server auto-deployment folder.
Check EJB CMP datasources	If this check box is selected, the names of persistent entities for EJB CMP data sources are validated. The validation is based on the mappings defined in WebLogic Server-specific EJB deployment descriptor files.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use or to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click –.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a local configuration

Item	Description
Run /	Use to switch between the settings for the run, debug and code coverage modes.

Debug /

Coverage

Startup script	Specify the script to be used to start the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Startup Script dialog which shows the contents of the Startup script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click … and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Specify the script to be used to stop the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Shutdown Script dialog which shows the contents of the Shutdown script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click … and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Pass environment variables	To pass specific variables to the server environment, select this check box and specify the variables: – To add a variable, click + and specify the variable name and value in the Name and Value fields respectively. – To remove a variable from the list, select the variable and click —.
Port	Use this field to change the debugger port.
Debugger Settings	Click this button to edit the debugger options on the Debugger page of the Settings dialog .

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

Item	Shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
—	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up /	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)

	Alt+Down	
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
−	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click . See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: WebSphere Server

Run | Edit Configurations | + | WebSphere Server | Local or Remote

WebSphere Server run/debug configurations let you deploy and debug your applications on WebSphere Application Server (version 6.1 or later) and the WebSphere Application Server Liberty profile. (The WebSphere Integration plugin must be enabled.)

When it's necessary to distinguish between the features for the Liberty profile and WebSphere Application Server, the first of the two is referred to as WAS LP, while the second - simply as WAS.

- Name field and Share option
- Server tab for a local configuration: WAS LP
- Server tab for a local configuration: WAS
- Server tab for a remote configuration: WAS LP
- Server tab for a remote configuration: WAS
- Deployment tab
- Logs tab
- Code Coverage tab
- Startup/Connection tab for a local configuration
- Startup/Connection tab for a remote configuration
- Before Launch options
- Toolbar

See also, Working with Server Run/Debug Configurations.

Name field and Share option

Item	Description
Name	Use this field to edit the name of the run/debug configuration. This field is not available when editing the run/debug configuration defaults.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.

Server tab for a local configuration: WAS LP

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (`Shift+Enter`) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`.
On 'Update' action	Select the necessary option for the Update application function (or `Ctrl+F10` in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.

Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Server	Select the name of the target server. If the list is empty, create a server: in your Liberty profile installation folder, from the `wlp/bin` directory, run the following command: `server create <server-name>` If the message *Error: JMX administration should be enabled* is shown in the lower part of the dialog, click Fix. As a result, the following is added into the configuration file `wlp/usr/servers/<server-name>/server.xml`: – `<feature>localConnector-1.0</feature>` (within `<featureManager>`) – `<applicationMonitor updateTrigger="mbean"/>`

Server tab for a local configuration: WAS

Item	Description
Application server	Select the server configuration to be used. Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after starting the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
VM options	If necessary, specify the command-line options to be passed to the server JVM at the server start. If you need more room to type, click next to the field to open the VM Options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The update options are different for exploded and packed artifacts. For exploded artifacts, the available options are: – Update resources. All changed resources are updated (HTML, JSP, JavaScript, CSS and image files). – Update classes and resources. Changed resources are updated; changed Java classes (EJBs, servlets, etc.) are recompiled. In the debug mode, the updated classes are hot-swapped. In the run mode, IntelliJ IDEA just updates the changed classes in the output folder. Whether such classes are actually reloaded in the running application, depends on the capabilities of the runtime being used. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed. For packed artifacts, the available options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed. – Restart server. The server is restarted. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Profile path	Select the server profile from the list. Alternatively, click ... and select the server profile directory in the dialog that opens .
Cell	Specify the target server cell.
Node	Specify the target server node.
Server	Specify the target server.
Username	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the server.
Password	The password of the user specified in the Username field .

Server tab for a remote configuration: WAS LP

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Server	Specify the name of the target server.
Test Connection	Click this button to test the connection with the server to make sure that the current settings are correct.
Host	The fully qualified domain name or the IP address of the server host.
Port	The server HTTP port.

Server tab for a remote configuration: WAS

Item	Description
Application server	Select the server configuration to be used. Note that this is a local server configuration. (When working with a remote server, the same server version must be available locally.) Click Configure to create a new server configuration or edit an existing one. (The Application Servers dialog will open.)
After launch	Select this check box to start a web browser after connecting to the server and deploying the artifacts. Select the browser from the list. Click ... (Shift+Enter) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
On 'Update' action	Select the necessary option for the Update application function (or Ctrl+F10 in the Run or Debug tool window). The options are: – Hot swap classes. Changed classes are recompiled and reloaded at runtime. This option works only in the debug mode. – Redeploy. The application artifact is rebuilt and redeployed.
Show dialog	Select this check box if you want to see the Update dialog every time you use the Update application function. The Update dialog is used to select the update option prior to actually updating the application.
On frame deactivation	Specify what IntelliJ IDEA should do when you switch from the IDE to a different application (for example, a web browser). (Frame deactivation means switching to a different application.) The options other than Do nothing have the same meanings as in the case of the Update action .
Cell	Specify the target server cell.
Node	Specify the target server node.
Server	Specify the target server.
Username	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the server.
Password	The password of the user specified in the Username field .
SOAP Port	Specify the server SOAP port.
Test Connection	Click this button to test the connection with the server to make sure that the current settings are correct.
Use SSL connection	Select this check box to connect to the server using SSL. Specify the associated settings: – Trust store. Specify the path to the truststore file. You can click ... and select the necessary file in the dialog that opens . – Trust store password. Specify the password for accessing the truststore. – Key store. Specify the path to the keystore file. You can click ... and select the necessary file in the dialog that opens . – Key store password. Specify the password for accessing the keystore.
Remote connection settings	The settings for accessing deployed applications.

Host	The fully qualified domain name or the IP address of the server host.
Port	The server HTTP port.

Deployment tab

Use this tab to specify which artifacts and/or external resources should be deployed onto the server. (An external resource means a deployable Web component such as a `.war` file which is not represented by a project artifact. Usually, such components are stored outside of the project scope.)

To add items to the deployment list (shown under Deploy at the server startup), use +. For more information, see the table below.

Note that deployment to WAS LP that runs on a different computer is not supported.

Item	Description
+(`Alt+Insert`)	Use this icon or shortcut to add an artifact or an external resource to the list. – To add an artifact, select Artifact and choose the desired artifact in the dialog that opens. – To add an external resource, select External Source and choose the location of the desired resource in the dialog that opens .
–(`Alt+Delete`)	Use this icon or shortcut to remove the selected artifacts and external resources from the list.
(`F4`)	Use this icon or shortcut to configure the selected artifact. (The Artifacts page of the Project Structure dialog will open.)
Use custom context root	If you want to assign a particular context root to an artifact or external resource, select the artifact or the resource, select the check box and specify the desired context root in the field underneath the check box.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the Run or Debug tool window .

Item	Description
Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: – Full paths to specific files. – Ant patterns that define the range of files to be displayed. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
+	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
–	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Code Coverage tab

Use this tab to configure code coverage monitoring options.

Note that this tab is not available for remote servers.

Item	Description
Choose code coverage runner	Select the desired code coverage runner.
Sampling	Select this option to measure code coverage with minimal slow-down.
Tracing	Select this option to collect accurate branch coverage. This mode is available for the IntelliJ IDEA code coverage runner only.
Track per test coverage	Select this check box to detect lines covered by one test and all tests covering line.
Packages and classes to record code coverage data	If necessary, specify the classes and packages to be measured. Use or to add classes or packages to the list. To remove the classes or packages from the list, select the corresponding list items and click –.
Enable coverage in test folders.	Select this check box to include the test source folders in code coverage analysis.

Startup/Connection tab for a local configuration

Item	Description

Run / Debug / Coverage	Use to switch between the settings for the run, debug and code coverage modes.
Startup script	Specify the script to be used to start the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Startup Script dialog which shows the contents of the Startup script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Shutdown script	Specify the script to be used to stop the server. Use default: – If this check box is selected, the default script is used. in this case opens the Default Shutdown Script dialog which shows the contents of the Shutdown script field (readonly). – Clear this check box to change the parameters passed to the script or to specify a different script: – To specify the script, click and select the desired script in the dialog that opens . – To specify the parameters, click and specify the script parameters and VM options in the Configure VM and Program Parameters dialog. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Pass environment variables	To pass specific variables to the server environment, select this check box and specify the variables: – To add a variable, click and specify the variable name and value in the Name and Value fields respectively. – To remove a variable from the list, select the variable and click .
Port	Use this field to change the debugger port.
Debugger Settings	Click this button to edit the debugger options on the Debugger page of the Settings dialog .

Startup/Connection tab for a remote configuration

This tab shows command-line options for starting the server JVM in the run and debug modes.

Item	Description
Run / Debug	Use to switch between the settings for the run and debug modes. The settings are shown in the area under To run/debug...
To run/debug remote server JVM...	The command-line options for starting the server JVM. These are shown just for copying elsewhere.
Transport (and all that follows)	The GUI for generating the remote debug command-line options shown in the area under To run/debug...

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

	Item	Shortcut	Description
+		Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to compile the project. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to build an artifact or artifacts. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run. – Run Ant target. Select this option to run an Ant target. In the dialog that opens, select the target to be run. For more information, see Ant . – Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to run a Maven goal. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External Tool. Select this option to run a remote application which is external to IntelliJ IDEA. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in IntelliJ IDEA yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools .
–		Alt+Delete	Click this icon to remove the selected task from the list.
			Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.

	Enter Alt+Up / Alt+Down	Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page		Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window		If this check box is selected, the Run or the Debug tool window opens when you start the run/debug configuration. Otherwise, the tool window isn't shown. However, when the configuration is running, you can open the corresponding tool window for it yourself if necessary.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Create a run/debug configuration.
−	Alt+Delete	Delete the selected run/debug configuration.
	Ctrl+D	Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
	Alt+Up / Alt+Down	Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use and . You can also drag a configuration into a folder. To remove grouping, select a folder and click −. See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: XSLT

This run/debug configuration becomes available, when XPath View+XSLT Support plugin is enabled.

On this page

- Getting Access to the Run/Debug Configuration: XSLT Dialog Box
- Settings Tab
- Advanced Tab
- Toolbar
- Common options

Getting Access to the Run/Debug Configuration: XSLT Dialog Box

Make sure the XPath View+XSLT Support plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Settings Tab

Item	Description
Input	Use the controls in this area to specify the XML file to process and the script to be executed. – XSLT Script File - in this text box, specify the path to the XSLT stylesheet file. Type the path manually or click the Browse button and select the desired file in the Choose XSLT File dialog box, that opens. – Choose XML Input File - from this drop-down list, select the XML input file to be transformed. The list contains all the XML files that have been associated with the chosen stylesheet via the File Associations functionality. To specify a file, which is not on the list, click the Browse button and select the desired file in the Choose XML File dialog box, that opens.
Output	Use the controls in this area to configure handling of the script output. – Show in Default Console - select this option to have the output displayed in the normal run console, together with any warnings and error messages from the XSLT transformer, as well as messages generated by the script, e.g. by `xsl:message` . – Show in Extra Console Tab - select this option to have the produced output displayed in an extra, XSLT Output, tab. This option is selected by default. – Highlight Output As - from this drop-down list, select the file type to highlight the output as. – Save to File - select this option to have the output saved directly to a file. In the text box, specify the name of the target file. Type the path to the file manually or click the Browse button and select the desired file in the Choose Output File dialog box, that opens. If you type the name of a file that does not exist, IntelliJ IDEA will create a file and save the output to it. – Open File in Editor After Execution - select this check box to have the file with the output opened in the editor after the script is executed successfully. – Open File in Web Browser After Execution - select this check box to have the file with the output opened in the configured Web browser after the script is executed successfully. Warning! The specified file will be overwritten without requesting for confirmation.
Parameters	Use the controls in this area to create and manage a list of parameters to be passed to the script. – Add (+) - click this button to create a new entry. – Remove (—) - click this button to remove the selected entry from the list. – Name - in this text box, specify the name of the parameter. – Value - in this text box, specify the value of the parameter. Warning! The field is mandatory. Parameters without values are not passed to the script. Values are not assigned by default.

Advanced Tab

In this tab, configure additional options that are not commonly required in run configurations.

Item	Description
Smart Error Handling	– Clear this check box to have the console display full error messages including their complete stack traces, when an error occurs during execution. – Select this check box to suppress showing staktraces and have the console display only the relevant information about errors.
VM Arguments	In this text box, specify optional VM arguments to be passed to the VM where the XSLT script is executed. These can be heap size, garbage collection options, file encoding, etc. If the line of VM arguments is too long, click the button and and type the text in the VM Arguments dialog box, that opens.
Working Directory	In this text box, specify the working directory to use. Type the path manually or click the Browse button and select the desired folder in the Working Directory dialog box, that opens. If no folder is specified in text box, the working directory will be the one where the XSLT script file is located.
Classpath and JDK	In this area, specify the environment to run the script in. By default, it is the module the XSLT file belongs to. – From Module - select this option to execute the script in a specific module. From this drop-down list, select the desired module. Tip The full classpath to the selected module is included. This can be required if the script uses custom XSLT Extension Functions. – Use JDK - select this option to choose the JDK without including anything module- or project-related into the classpath. Tip It can be useful to explicitly choose a specific JDK to test the script with.

Toolbar

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new configuration to the list.
—	Alt+Delete	Click this button to remove the selected configuration from the list.
	Ctrl+D	Click this button to create a copy of the selected configuration.

	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
or	Alt+Up or Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

Item	Description
Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise. This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, IntelliJ IDEA checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
+	Alt+Insert	Click this icon to add a task to the list. Select the task to be added: – Run External tool. Select this option to run an application which is external to IntelliJ IDEA. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in IntelliJ IDEA yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools . – Make. Select this option to have the project or module compiled. The Make Module command will be carried out if a particular module is specified in the run/debug configuration, and the Make Project command otherwise. If an error occurs during the compilation, IntelliJ IDEA won't attempt to start the run/debug configuration. – Make, no error check. The same as the Make option but IntelliJ IDEA will try to start the run/debug configuration irrespective of the compilation result. – Build Artifacts. Select this option to have an artifact or artifacts built. In the dialog that opens, select the artifact or artifacts that should be built. See also, Working with Artifacts . – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run Ant target. Select this option to have an Ant target run. In the dialog that opens, select the target to be run. For more information, see Ant . – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support . – Run Maven Goal. Select this option to have a Maven goal run. In the dialog that opens, select the goal to be run. For more information, see Maven . – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
–	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.

	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run / Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Save Project As Template dialog

Tools | Save Project As Template

Specify the settings for the template that you are creating.

Item	Description
Save	If the project that you are saving as a template contains more than one module, you can select to create a template for the whole project, or to include only for one of the modules in the template. So select <whole project> or the name of the corresponding module. This option is not available if your project contains only one module.
Name	Specify the name of the template.
Description	Type the template description. You can enclose text fragments between `<b>` and `</b>`, and `<i>` and `</i>` to make the corresponding fragments bold or italicized. For example: `A project with a <b>Java</b> module and support for <i>Web development</i> and <i>Tomcat</i>.`
Replace parameters with placeholders	If you select this check box (recommended), you will have additional options when creating a new template-based project or module. For example, you will be able to specify the base Java package and an application server different from the ones defined in the current project.

Select Path Dialog

Use this dialog to select the necessary files or folders.

The dialog name and the available functions depend on the task you are performing at the moment (inappropriate functions are normally disabled). For example, depending on the situation, you may be able to select only one item, or a number of items. There may be cases when you can select a folder or folders but cannot select a file or files, etc.

– Main functions
– Path field
– Using drag-and-drop from a file browser

Main functions

Most of the functions available in this dialog are accessed by means of the toolbar icons (shown in the Icon column). Alternatively, you can use context menu commands (accessed by right-clicking items in the tree; listed in the Command column) or keyboard shortcuts (the Shortcut column).

Icon	Command	Shortcut	Description
	Home	Ctrl+1	Use this icon, command or shortcut to select your home directory. For example, on Windows, this may be `C:\Users\<your_name>`.
	Desktop	Ctrl+D	Use this icon, command or shortcut to select the Desktop directory
	Project	Ctrl+2	Use this icon, command or shortcut to select your project root directory.
	Module	Ctrl+3	Use this icon, command or shortcut to select the root directory of the current module.
	New Folder	Alt+Insert	Use this icon, command or shortcut to create a new folder in the selected folder.
	Delete	Delete	Use this icon, command or shortcut to delete the selected file or folder.
	Refresh	Ctrl+Alt+Y	Use this icon, command or shortcut to synchronize the tree with the current state of the file system. (Under certain circumstances, IntelliJ IDEA may not be aware of the changes made externally unless you use this command.)
	Show or Hide Hidden Files and Folders		Use this icon or command to turn showing hidden files and folders on or off.
	Show Quick List		Use this icon or command to show a list of all JDKs available on your computer. This icon is only available for selecting JDK.
	Hide or Show path	Ctrl+P	Use this command or shortcut to hide or show the path field. (The command is located on the toolbar in the right-hand part of the dialog and is shown as a hyperlink.)

Path field

The path field (if not hidden) is located underneath the toolbar. This field shows the path to the item selected in the tree.

By using the path auto-completion feature available in this field, you can quickly navigate through the file system to select the necessary file or folder.

To activate path auto-completion, place the cursor in the field and press Ctrl+Space. Start typing. A pop-up will appear showing the contents of the current directory. Select an item in the pop-up. Continue typing and selecting until the necessary item is selected.

Use the button to the right to show the history list of recent entries.

Using drag-and-drop from a file browser

You can quickly locate and select the necessary file or folder if you drag the corresponding item from your file browser (Explorer, Finder, etc.) into the area where the tree is shown.

Setup Library dialog

Use this dialog to set up a library .

- If the necessary files (`.jar`) are already available on your computer, you can arrange those files in a library and use that new library. To do that, select Use library, click Create and select the necessary files in the dialog that opens . (Use the `Ctrl` key for multiple selections.)
  Optionally, click Configure to edit your new library. (The Create Library Dialog will open.)
- You can also download the necessary files and use those files as a library. To do that, select Download.
  Optionally, click Configure to edit the library settings and contents. (The Downloading Options Dialog will open.)

Show History for File / Selection Dialog

VCS | Local History | Show History or Show History for Selection

Use this dialog to explore changes to a file, or selection. There are two views in this dialog:

- History view in the left-hand part
- Differences view in the right-hand part

Tip The same dialog boxes are available on the context menu of a file or selected text in the editor.

History view

This view shows the list of revisions (states) of a file, with the date and time when the revision was stored. Some of the revisions are supplied with tags and labels.

Revisions are tagged automatically, for example, on opening a project, committing changes, or performing test. You can also set your own labels .

Item	Description
	Click this button to revert the selected action.
	Click this button to create a patch based on the selected local version.
	Click this button to open the corresponding help topic.

Tip The same actions are available on the context menu of each revision.

Differences view

The Differences view is a powerful editor that supports basic search and replace , undo/redo actions , and code completion .

If a revision is selected in the History view , the left-hand pane of the Differences view shows this read-only revision, with the differences against the current revision which is displayed in the right-hand pane. The current revision can be edited.

Item	Shortcut	Description
	Ctrl+C	Click this button to copy the current line or the selected fragment to the clipboard.
	Ctrl+F Ctrl+R	Click this button to initiate the finding and replacing text procedure in the pane where the caret currently resides. Refer to the search options description for details.
or	F7 or Shift+F7	Use these buttons to move to the next or previous difference.
Ignore whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text. – Do not ignore - when this option is selected, white spaces are considered unimportant and the differences are highlighted. – Leading and Trailing - select this option to have differences in the end and in the beginning of a line ignored. – All - when this option is selected, white spaces are considered unimportant regardless of their location in the source code.
or		Use these buttons to apply differences.
Legend		This area shows summary information about the encountered differences: the number of differences found and the color map. The color map for the Differences viewer is configured on the Colors and Fonts page .

Show History for Folder Dialog

VCS | Local History | Show History

Use this dialog to explore changes to a folder. There are two views in this dialog:

- History view in the left-hand part
- Changes view in the right-hand part

Tip The same dialog box is available on the context menu of a folder.

History view

The History view shows a list of folder revisions (states), each one being supplied with a time stamp, revision number, and an indication of the action that resulted in that state.

Item	Description
	Click this button to revert the selected action.
	Click this button to create a patch based on the selected local version.
	Click this button to open the corresponding help topic.

Changes view

The Changes view shows the differences between the current state and the one selected in the History view . The differences are shown as a tree of changed (new, modified and deleted) files and subfolders.

Item	Shortcut	Description
	Ctrl+D	Click this button to show the differences between the current local version and the one selected in the History view. Alternatively, double-click the current local version in the Changes view.
		With a file selected in the Changes view, click this button to roll back the selected action.
	Ctrl+P	Click this button to show changed files as a tree view of folders. If this button is not pressed, the files are shown as a flat list.
or	Ctrl+NumPad Plus or Ctrl+NumPad -	Click these buttons to have all nodes expanded or collapsed. These buttons are only available when the changed files are shown as a tree view.
	Ctrl+A	Click this button to select all the files in the list or a tree view.
	Ctrl+F	In this area, type the search string. Note also that speed search is available in the Changes pane.
		Click this button to clear the search area.

Tip Speed search is available in the Version Control view.

Specify Dependency Analysis Scope Dialog

Analyze | Analyze Dependencies

Analyze | Analyze Backward Dependencies

Analyze | Analyze Dependency Matrix

Analyze | Infer Nullity

Use this dialog box to define the scope for the search of dependencies or nullable elements. The contents of the dialog boxes slightly differ depending on the type of analysis.

Item	Description
Whole project	Select this option to perform the analysis for the whole project.
Module <name>	Select this option to have IntelliJ IDEA analyze the module that is currently selected in the Project tool window.
File <name>	Select this option to analyze the file that is currently selected in the Project tool window or opened in the editor.
Selected files	Select this option to analyze the files that are currently selected in the Project tool window.
Uncommitted files	This scope is only available for the projects under version control. Select this option to have IntelliJ IDEA analyze only files that have not been committed to the version control system. Use the drop-down list to further limit the analysis scope. The available options are: – All - select this option to have files from all changelists analyzed. – Default - select this option to have IntelliJ IDEA analyze only files from the Default changelist.
Custom scope	Select this option to use a custom scope. Select a pre-defined scope from the drop-down list, or click ... and define the scope in the Scopes dialog . Tip Use a special language to define a scope.
Include test sources	Select this check box to perform analysis on the test sources.
Scope of interest	From this drop-down list, select the scope to seek for backward dependencies. Tip The field is available for the backward dependencies analysis only.
Show transitive dependencies. Do not travel deeper than	Select this check box to have IntelliJ IDEA analyze transitive dependencies. From the Do not travel deeper than drop-down list, choose the desired level. Tip The field is available for the dependencies analysis only.
Annotate local variables	If this check box is selected, then the local variables of a class will be included in the nullity analysis, and annotated. Tip The field is available for inferring nullity only.

Specify Code Duplication Analysis Scope

Analyze | Locate Duplicates

Use this dialog box to launch the search for duplicated code fragments in the specified scope.

Item	Description
Whole project	Select this option to perform the analysis for the whole project.
Module <name>	Select this option to have IntelliJ IDEA analyze the module that is currently selected in the Project tool window.
File <name>	Select this option to analyze the file that is currently selected in the Project tool window or opened in the editor.
Selected files	Select this option to analyze the files that are currently selected in the Project tool window.
Uncommitted files	This scope is only available for the projects under version control. Select this option to have IntelliJ IDEA analyze only files that have not been committed to the version control system. Use the drop-down list to further limit the analysis scope. The available options are: – All - select this option to have files from all changelists analyzed. – Default - select this option to have IntelliJ IDEA analyze only files from the Default changelist.
Custom scope	Select this option to use a custom scope. Select a pre-defined scope from the drop-down list, or click ... and define the scope in the Scopes dialog . Tip Use a special language to define a scope.
Include test sources	Select this check box to perform analysis on the test sources.

Specify Code Cleanup Scope Dialog

Analyze | Code Cleanup

Specify the scope for code cleanup. (Code cleanup means finding potentially problematic code fragments and automatically fixing them right away.)

Item	Description
Whole project	Select this option if you want to perform code cleanup for the whole project.
Uncommitted files	This option is only available for projects under version control. Select this option if you only want to perform code cleanup for the files that have not yet been committed to a version control system, and choose a changelist from the drop-down list.
File <file path>	Select this option to perform code cleanup for the file that is open in the editor or selected in the Project tool window.
Module <module name>	Select this option to perform code cleanup for the module that is currently selected in the Project tool window. This option is only available, when a IntelliJ IDEA project consists of several modules .
Directory <directory path>	Select this option to perform code cleanup for the directory currently selected in the Project tool window.
Selected files	Select this option to perform code cleanup for the files selected in the Project tool window.
Custom scope	Select this option to specify a custom scope. Select one of the predefined scopes from the drop-down list, or click ... and define the scope in the Scopes dialog that opens. For instructions on how to define a scope, refer to Scope Language Syntax Reference .
Include test sources	Select this option to apply code cleanup to test sources.
Inspection profile	Select the inspection profile to be used. Choose a pre-defined profile from the drop-down list, or click ... and configure a profile in the Inspections dialog that opens. You can open the Inspections dialog to check which fixes will be applied to the selected scope when you run code cleanup.

Specify Inspection Scope Dialog

Analyze | Inspect Code

Use this dialog box to define the scope to apply inspection to and the profile against which the source code should be inspected.

Note that the list of scopes varies depending on the project type.

Item	Description
Whole project	Select this option to perform the analysis for the whole project.
File <name>	Select this option to analyze the file that is currently selected in the Project tool window or opened in the editor.
Selected files	Select this option to analyze the files that are currently selected in the Project tool window.
Uncommitted files	This scope is only available for the projects under version control. Select this option to have IntelliJ IDEA analyze only files that have not been committed to the version control system. Use the drop-down list to further limit the analysis scope. The available options are: − All - select this option to have files from all changelists analyzed. − Default - select this option to have IntelliJ IDEA analyze only files from the Default changelist.
Custom scope	Select this option to use a custom scope. Select a pre-defined scope from the drop-down list, or click ... and define the scope in the Scopes dialog . Tip Use a special language to define a scope.
Include test sources	Select this check box to perform analysis on the test sources.
Inspection profile	Select a profile to inspect the specified scope against. A profile is selected from the drop-down list. If the desired profile is not in the list, click the ellipsis button and configure the desired profile on the Inspections page of the Settings dialog.

Structural Search and Replace Dialogs

Edit | Find | Search Structurally

Edit | Find | Replace Structurally

Use these dialog boxes to find and replace fragments of code that structurally match the suggested search template .

Tip To learn more about the possible usages, refer to the section Structural Search and Replace .

Item	Description	Available in
Search template	Use this text area to specify the template to be sought for. You can type the template code in the field or click the Copy existing template button to use one of the existing templates.	Both
Replacement template	Use this text area to specify the template to be substituted. You can type the template code in the field or click the Copy existing template button to use one of the existing templates.	Structural Replace
Save template	Click this button to open the Save Template dialog box, where you have to specify the name of the new template. Note that the new template is stored under the User Defined node of the existing templates tree view.	Both
Edit variables	Calls the Edit variables dialog box to set constraints for template variables.	Both
History	Click this button to open the History dialog box, that shows up to 25 last invoked templates.	Both
Copy existing template	Click this button to open the Existing Templates dialog box, where you can select one of the pre-defined or custom templates. Selected template is displayed in the Preview field. Clicking OK in the Existing Templates dialog box inserts the source code of the template into the Search template or Replace template field.	Both
Recursive matching	If this check box is selected, the search is performed recursively in the results.	Structural Search
Case sensitive	If this check box is selected, the search discerns lower and upper case letters.	Both
File type	Select file type from the drop-down list.	Both
Shorten fully qualified names	This option makes sense in case the template text contains fully qualified class names. If the check box is selected, IntelliJ IDEA automatically reduces these names in the template. Otherwise, fully qualified class names are used.	Structural Replace
Reformat according to style	Check this option, if you want IntelliJ IDEA to automatically reformat the expanded code fragment according to your code style settings (for details, refer to the Code Style dialog box). If the option is not checked, IntelliJ IDEA will only indent the whole template according to the position in code at which it is expanded, leaving its formatting as is.	Structural Replace
Use static import if possible	Check this option, if you want IntelliJ IDEA to shorten any references to static elements in the replaced code. IntelliJ IDEA will insert a static import for those elements. The elements are then referenced by their short name. If there are no references to static elements in the replaced code, the option will be ignored.	Structural Replace
Scope	Select one of the existing scopes from the drop-down list or click the ellipsis button (alternatively, press `Shift+Enter`), and create your own scope in the Scopes dialog box.	Both
Open in new tab	If this check box is selected, the results of the new search display in a new tab in the Find results tool window. Otherwise, the search results update the existing tab.	Both

Structural Search and Replace. Edit Variable Dialog

Edit | Find | Search Structurally | Edit variables

Use this dialog to define constraints for the variables of a search template .

Tip The contents of the dialog box depend on the selected variable type.

Item	Description
Variables	This area shows a list of variables used in the current search template.
Text constraints	In this area define the following constraints of the selected variable regarding text: – Text/regular expression - in this text box, type a perl-like expression or a class name to be used as a variable constraint. Basic code completion is available for class names. – Invert condition - select this check box to have the text pattern inverted. – Apply constraint within type hierarchy - select this check box to have the search according to the pattern performed both in type names and in parents (within the hierarchy). – Whole words only - when this check box is selected, only whole words within text are matched. This option recognizes string literals and comments.
Occurrences count	In this area, define how pattern hits will be counted. – Minimum count - in this text box, type the minimum number of elements in the list. – Maximum count - in this text box, type the maximum number of elements in the list. – Unlimited - select this check box to allow unlimited number of elements in the list.
Expression constraints	In this area, define how expressions should be processed. – Value is read - if this check box is selected, the matching variable is to be read. – Value is written - if this check box is selected, the matching variable is to be written. – Expression type (regexp) - if the calculated variable is an expression, this constraint checks its type. For instance, for the `foo($a$)` expression the type of the method parameter would be checked. – Expected type of expression (regexp) - if the calculated variable is matched to any expected type of an expression, this constraint checks the expression type anywhere the expression was used. For instance, correspondence between the method parameter type (e.g. `$a$`) in method calls will be checked for methods like `foo($a$)` . – Apply constraint within type hierarchy - select this check box to have the search according to the pattern performed both in type names and in parents (within the hierarchy). – Invert condition - select this check box to have the value of the corresponding check box changed to the opposite one.
Script constraints	In this area, define a variable constraint via a script. Specify the script in the text box or click the ... button to open the Edit Groovy Script Constraint dialog box. The constraint is applied after the initial matching process is finished.
This variable is target of the search	If this check box is selected, the search results will show not the entire expression but the selected variable(s) only.

Edit Variables. Complete Match Dialog

Use the Complete Match dialog to define constraints for the entire pattern that you have specified in the search template .

Item	Description
Text constraints	In this area define the constraints of the selected variable regarding text. In case of Complete Match this area might not be useful. The constraints are as follows: – Text/regular expression - in this text box, type a perl-like expression or a class name to be used as a variable constraint. Basic code completion is available for class names. – Invert condition - select this check box to have the text pattern inverted. – Apply constraint within type hierarchy - select this check box to have the search according to the pattern performed both in type names and in parents (within the hierarchy). – Whole words only - when this check box is selected, only whole words within text are matched. This option recognizes string literals and comments.
Contained in constraints	In this area, define additional constraint pattern inside the already defined search template. Specify the pattern in the text box or click the … button to open the Existing Templates dialog box. Invert condition - select this check box to have the value of the corresponding field changed to the opposite one.
Script constraints	In this area, define a variable constraint via a script. Specify the script in the text box or click the … button to open the Edit Groovy Script Constraint dialog box. The constraint is applied after the initial matching process is finished.
This variable is target of the search	If this check box is selected, the search results will show not the entire expression but the selected variable(s) only.

Type Migration Dialog

Refactor | Type Migration

Ctrl+Shift+F6

Item	Description
Migrate type	Use this drop-down list to specify the new type.
Choose scope	Use this drop-down list to specify the migration scope. Click the Browse button ..., if necessary.
Refactor	Click this button to launch refactoring to all usages in the specified scope. If there are any conflicts, IntelliJ IDEA will notify you about them.
Preview	Click this button to open Type Migration Preview where you can browse items to be changed, exclude/include them from refactoring, and view the conflicts detected.

Web Server Debug Validation Dialog

Run | Web Server Debug Validation

Use this dialog box to initiate validation of the debugger setup in a local or remote environment. In the context of IntelliJ IDEA, the term remote denotes any environment outside the project root. This can be a server on physically remote host, or a Vagrant box, or a server on the same machine but in a folder outside your project root.

Item	Description
Local Web Server or Shared Folder	Choose this option to check a debugger associated with a local Web server. – Path to Create Validation Script: In this field, specify the absolute path to the folder under the server document root where the validation script will be created. For Web servers of the type Inplace, the folder is under the project root. The folder must be accessible through http. – URL to Validation Script: In this text box, type the URL address of the folder where the validation script will be created. If the project root is mapped to a folder accessible through `http`, you can specify the project root or any other folder under it.
Remote Web Server	Choose this option to check a debugger associated with a remote server. – Path to Create Validation Script: In this field, specify the absolute path to the folder under the server document root where the validation script will be created. The folder must be accessible through http. – Deployment Server: In this field, specify the server access configuration of the type Local Server or Remote Server to access the target environment. For details see Configuring Synchronization with a Web Server . Choose a configuration from the drop-down list or click the Browse button ... in the Deployment dialog .
Validate	Click this button to have IntelliJ IDEA create a validation script, deploy it to the target remote environment, and run it there.

Settings / Preferences Dialog

File | Settings for Windows and Linux

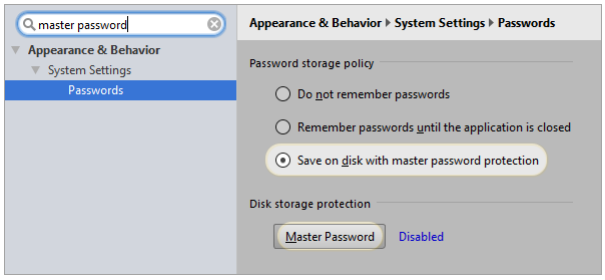
IntelliJ IDEA | Preferences for OS X

`Ctrl+Alt+S`

Note The settings that pertain to the current project, are marked with the icon.

The Settings dialog lets you control every aspect of the IntelliJ IDEA behavior and appearance.

Use the search box in the upper-left part of the dialog to find the options of interest. Alternatively, you can browse the settings using the hierarchical list of categories (groups of settings) underneath the search box.



On this page find the descriptions of the main controls of the dialog.

Item	Description
Search	Enter a search keyword in the text area. While typing the search string, the list of options in the dialog reduces to the matching occurrences.
	Click this button to clear the search area.
OK	Apply changes and close the dialog box.
Cancel	Discard changes and close the dialog box.
Apply	Apply changes and leave the dialog box opened.
Help	Show reference page.

- Appearance and Behavior
- Keymap
- Editor
- Plugins
- Version Control
- Build, Execution, Deployment
- Languages and Frameworks
- Tools

Appearance and Behavior

File | Settings | Appearance and Behavior for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior for OS X `Ctrl+Alt+S`

When you select the Appearance and Behavior category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- Appearance
- Menus and Toolbars
- System Settings
- File Colors
- Scopes
- Notifications
- Quick Lists
- Path Variables

Appearance

File | Settings | Appearance and Behavior | Appearance for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | Appearance for OS X

`Ctrl+Alt+S`

Use this page to change the overall look and feel of your IDE.

– UI Options
– Antialiasing
– Window Options
– Presentation Mode

UI Options

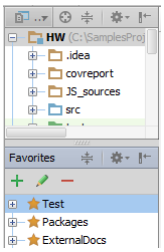
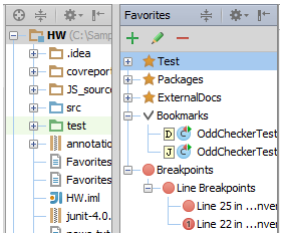
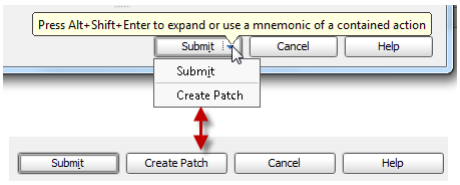
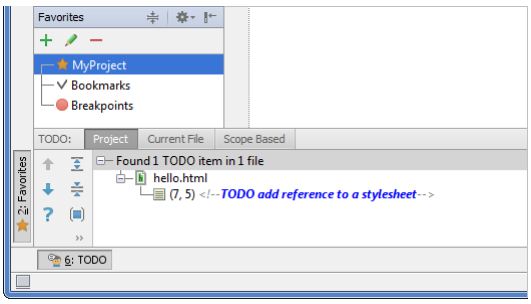
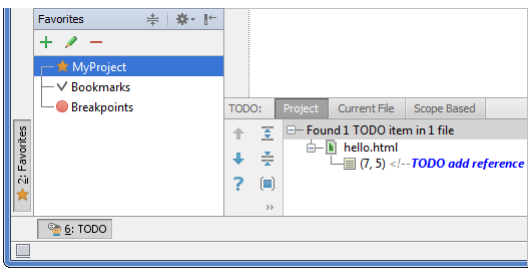
Option	Description
Theme	Use this drop-down list to select the desired theme from the list. In particular, Darcula theme is available. Changing to/from this theme requires IntelliJ IDEA restart. In the Community edition of IntelliJ IDEA, the new default scheme with the name IntelliJ is used.
Adjust colors for red-green vision deficiency	Select this option to adjust the IDE colors (code highlighting in the editor, text notifications, etc.) for people with the red-green color deficiency. For more information, see Color-Deficiency Adjustment .
Override default fonts by (not recommended)	Select this check box to enable specifying font family and size to be used instead of the default one. When first installed, IntelliJ IDEA takes Windows default font size and style.
Cyclic scrolling in list	Select this check box to enable scrolling through a list by jumping from the last item to the first one and vice versa.
Show icons in quick navigation	Select this check box to have icons shown in the quick navigation pop-up menu (`Ctrl` / `Ctrl+Shift` / `Ctrl+Shift+Alt+N`) .
Automatically position mouse cursor on default button	Select this check box to have the mouse pointer placed at the default button when a dialog box opens. If the check box is not selected, the pointer location does not change.
Hide navigation popups on focus loss	If this check box is selected, the navigation pop-up frames (go to class/file/symbol) close, when any other IntelliJ IDEA component gets the focus. If this check box is not selected, the navigation pop-up frames persist on changing the focus, and the only way to close such pop-up lays with pressing `Escape` .
Drag-n-Drop with ALT pressed only	If this check box is not selected (by default), IntelliJ IDEA allows moving editor tabs, tool window buttons, files and folders in the Project tool window , using drag-n-drop. Select this check box to avoid accidental moving of a file or folder, or a UI component. Thus drag-n-drop only works while ALT key is pressed.
Tooltip initial delay (ms)	Use this slider to specify the time to pass between the moment you hover the mouse over an item in the editor and the moment when the tooltip with its value appears. This settings is especially important during debugging. If the delay is too short using the mouse becomes inconvenient because every mouse move across the screen brings forward a number of tooltips with the values of all the variables.

Antialiasing

Option	Description
IDE	From this drop-down list, select which antialising mode you want to apply to the IDE (including menus, tool windows, etc.): – Subpixel: this option is recommended for LCD displays and takes advantage of the fact that each pixel on a colour LCD is composed of red, green and blue sub-pixels. This allows smoothing text and rendering it with greater detail. – Greyscale: this option is recommended for non-LCD displays or displays positioned vertically. It deals with text at the pixel level. – No antialiasing: this option can be used for displays with high resolution, where non-antialiased fonts are rendered faster and may look better.
Editor	From this drop-down list, select which antialiasing mode you want to apply to the Editor : – Subpixel: this option is recommended for LCD displays and takes advantage of the fact that each pixel on a colour LCD is composed of red, green and blue sub-pixels. This allows smoothing text and rendering it with greater detail. – Greyscale: this option is recommended for non-LCD displays or displays positioned vertically. It deals with text at the pixel level. – No antialiasing: this option can be used for displays with high resolution, where non-antialiased fonts are rendered faster and may look better.

Window Options

Option	Description
Animate windows	Select this check box to have undocked tool windows slide with the animation effect. This option applies only when a tool window is undocked.
Show memory indicator	Select this check box to show the Memory Indicator on the Status Bar .
Disable mnemonics in menu	Select this check box to hide underlining of hot keys in the IntelliJ IDEA menus.
Disable mnemonics in controls	Select this check box to hide underlining of hot keys in the IntelliJ IDEA controls.
Display icons in	If this check box is selected (by default), the icons are displayed to the left of the menu commands.

menu items	If this check box is not selected, the menu commands are displayed without icons.
Side by side layout on the left/right	When these check boxes are selected, the way the tool windows are positioned is optimized for a wide-screen display. Side-by-side layout is OFF:  Side by side layout is ON:  Toggle layout by `Ctrl+MouseClick` on splitter between the tool windows. This only applies to the tool windows located on the left and right sides, but not at the top and bottom of the IntelliJ IDEA window.
Show tool window bars	Select this check box to display tool window bars.
Show tool window numbers	Select this check box to show tool window quick access numbers on the tool window buttons. You can use the `Alt+number` shortcuts regardless of this setting and change the shortcuts on the Keymap page . Note that the tool window mnemonics show up only when the corresponding keybindings have the format `Alt+n` , where `n` is an integer number in the range from 1 to 9. In the case of a different keyboard shortcut, the mnemonics are not displayed.
Allow merging buttons on dialogs	If this check box is selected, the multiple commands in a dialog box are grouped under a single button with a down arrow. You can view all merged commands by clicking the drop-down list, or pressing `Shift+Alt+Enter` . If this check box is not selected, the buttons will be shown in a row. Compare: 
Small labels in editor tabs	If this check box is selected, the font size on the editor tabs is set to the smaller value. If this check box is not selected, the font size on the editor tabs is set to the default value, as in the project tree view.
Widescreen tool window layout	If this check box is selected, the way the tool windows are positioned is optimized for a wide-screen display. Widescreen tool window layout is OFF:  Widescreen tool window layout is ON: 

Presentation Mode

ItemDescription

Font size	Use the drop-down list to select the font size for the presentation mode .

Menus and Toolbars

File | Settings | Appearance and Behavior | Appearance | Menus and Toolbars for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | Appearance | Menus and Toolbars for OS X

`Ctrl+Alt+S`

Use this page to configure the IntelliJ IDEA menus and toolbars .

In this topic:

– Menus and Items List
– Controls

Menus and Items List

The list shows the items for the menus and toolbars. The items are grouped according to the areas of their use.

To configure an item, expand the corresponding node and select the desired item. After that, the buttons in the right-hand part of the page become available.

Controls

Item	Description
Add After	Click this button to add a new action to the menu after the selected one. In the Choose Actions to Add dialog box that opens choose the desired action and optionally assign an icon to it.
Add Separator	Click this button to have a separator added to the menu after the selected item.
Edit Action Icon	Click this button to associate an icon with the selected menu item. In the Choose Action Icon Path dialog box that opens specify the path to the desired image. Tip – The image file should have `.png` extension. – The size of the toolbar icons should be 16x16.
Remove	Click this button to delete the selected item from the list.
Move Up	Click this button to move the selected item one position up.
Move Down	Click this button to move the selected item one position down.
Restore All Defaults	Click this button to abandon all the changes made to the all items and return to the default settings.
Restore Default	Click this button to abandon all the changes made to the selected item and return to the default settings.

Choose Actions to Add Dialog

The dialog box opens when you select an item in the Menus and Items List and click the Add After button.

In the dialog box, choose the desired action to be added to the menu or toolbar and optionally assign an icon to it.

Item	Description
Action List	The list shows all the actions available in IntelliJ IDEA. The actions are grouped below nodes according to the areas of their use.
Icon Path	In this text box, specify the location of the file with the icon you want to assign to the selected action. If necessary, use the Browse button ... to select the file in the corresponding dialog . Tip – The image file should have .png extension. – The size of the toolbar icons should be 16x16.
Set Icon	Click this button to associate the selected action with the icon specified in the Icon Path dialog box.

System Settings

File | Settings | Appearance and Behavior | System Settings

IntelliJ IDEA | Preferences | Appearance and Behavior | System Settings

Use this page to configure general behavior of IntelliJ IDEA.

Startup/Shutdown

Item	Description
Reopen last project on startup	Select this check box to have IntelliJ IDEA re-open the last opened project on startup.
Confirm application exit	Select this check box to have a warning message displayed when you attempt to close IntelliJ IDEA.

Project opening

Item	Description
Open project in a new window	Click this radio button to always open a new project in a new window.
Open project in the same window	Click this radio button to always close the current project, and reuse the same window.
Confirm window to open project in	Click this radio button to have IntelliJ IDEA ask you whether you want to open a new project in the same frame, or in a new one.

Synchronization

Item	Description
Synchronize files on frame or editor tab activation	If this check box is selected, all the files that were changed externally are reloaded from disk when you switch to IntelliJ IDEA from a different application, or when you you switch to their editor tab.
Save files on frame deactivation	If this check box is selected, all modified files are auto saved when you switch from IntelliJ IDEA to a different application. Note that you cannot disable autosave completely by turning off this and the following option. See Saving and Reverting Changes .
Save files automatically if application is idle for N seconds	If this check box is selected, all modified files are auto saved at regular time internals. See also, Saving and Reverting Changes .
Use "safe write" (save changes to a temporary file first)	If this check box is selected, a changed file is first saved in a temporary file. If the save operation succeeds, the file being saved is replaced with the saved file. (Technically, the original file is deleted and the temporary file is renamed.) Also, the ownership of such file changes. If this check box is not selected, the ownership of a file does not change, but all the advantages of safe write will be lost.

Passwords

File | Settings | Appearance and Behavior | System Settings | Passwords for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | System Settings | Passwords for OS X

Ctrl+Alt+S

Specify whether IntelliJ IDEA should remember or save your passwords - ones you use to access password-protected resources such as version control repositories.

If necessary, set a master password. If you do so, instead of the individual passwords - if they are stored in the IntelliJ IDEA password database - you'll be able to use your master password.

Note that the password storage policy defined on this page doesn't affect your database passwords. Whether those passwords should be saved is specified for each of the passwords separately, in the settings for the corresponding data sources. See, for example, DB data source settings .

- Password storage policy
- Disk storage protection

Password storage policy

Select one of the following options:

- Do not remember passwords. IntelliJ IDEA will ask you for a password each time it needs to access the corresponding password-protected resource.
- Remember passwords until the application is closed. IntelliJ IDEA will ask you for a password only the first time it needs to use that password. Then, until you quit the IDE, you won't be asked for that password again.
- Save on disk with master password protection. The passwords will be stored encrypted in the IntelliJ IDEA password database. To access this database, a master password will be used.
  Instead of the passwords themselves, IntelliJ IDEA will ask you for the master password. For each of the passwords stored in the database, this will happen only once during each session.

Disk storage protection

Master Password. Click this button to set or change the master password. (The Setup Master Password or the Change Master Password dialog will open.)

The master password is used to access the IntelliJ IDEA password database and also to strengthen the encryption of the stored passwords.

Enabled/Disabled. The text to the right of the Master Password button reflects the state of the master password. Enabled means that the master password is set and is not empty. Disabled means that the password is not set or is empty.

Reset Master Password Dialog

This dialog opens when you click Reset in the Enter Master Password or the Change Master Password dialog .

Warning! If you reset the master password, all your passwords currently stored in the password database will be lost.

Specify your new master password.

You can set the password empty. As a result, IntelliJ IDEA won't ask you for the password when accessing the password database. However, the security of your passwords in that case will be lower.

To come back to the previous dialog, click Enter or Change .

Item	Description
New password	Specify your new master password.
Confirm password	Type the new master password once more.
Encrypt with OS user credentials	For Windows only: if you select this check box, the master password itself will be stored encrypted in the password database. (Your Windows logon credentials will be used for encryption.) As a result, IntelliJ IDEA won't ask you for the master password when accessing the password database.
Enter or Change	If you decided not to reset the password (e.g. you remembered it), click this button to come back to the previous dialog. This may be the Enter Master Password or the Change Master Password dialog .

Setup Master Password Dialog

If your master password is not set yet, you can open this dialog by clicking Master Password on the Passwords page . In certain circumstances, this dialog may also open automatically.

Specify the password that you want to be set as your master password. (The master password is used to access the IntelliJ IDEA password database where - if so specified - your passwords are stored. Instead of the individual passwords, you'll be able to use your master password.)

You can set the password empty. As a result, IntelliJ IDEA won't ask you for the password when accessing the password database. However, the security of your passwords in that case will be lower.

Item	Description
Password	Specify the master password.
Confirm password	Type the master password once more.
Encrypt with OS user credentials	For Windows only: if you select this check box, the master password itself will be stored encrypted in the password database. (Your Windows logon credentials will be used for encryption.) As a result, IntelliJ IDEA won't ask you for the master password when accessing the password database.

Change Master Password Dialog
File | Settings | Passwords - Master Password for Windows and Linux

IntelliJ IDEA | Preferences | Passwords - Change Master Password for OS X

Ctrl+Alt+S

If your master password is already set, you can open this dialog by clicking Master Password on the Passwords page .

Specify your current master password, and then a new master password. (The master password is used to access the IntelliJ IDEA password database where - if so specified - your passwords are stored.)

You can set the new password empty. As a result, IntelliJ IDEA won't ask you for the password when accessing the password database. However, the security of your passwords in that case will be lower.

If you don't remember the current password, you can reset the master password (click Reset).

Item	Description
Password	Specify the current master password.
New password	Specify your new master password.
Confirm password	Type the new master password once more.
Encrypt with OS user credentials	For Windows only: if you select this check box, the master password itself will be stored encrypted in the password database. (Your Windows logon credentials will be used for encryption.) As a result, IntelliJ IDEA won't ask you for the master password when accessing the password database.
Reset	If you don't remember the current master password, click this button to reset it. (The Reset Master Password dialog will open.)

Enter Master Password Dialog

This dialog opens automatically when IntelliJ IDEA needs to access its password database to retrieve the necessary password.

Specify your master password.

If you don't remember the password, you can reset it (click Reset).

Item	Description
Password	Specify the master password.
Reset	If you don't remember the master password, click this button to reset it. (The Reset Master Password dialog will open.)

HTTP Proxy

File | Settings | Appearance and Behavior | System Settings | HTTP Proxy for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | System Settings | HTTP Proxy for OS X

Ctrl+Alt+S

If to access the Internet IntelliJ IDEA should use an HTTP proxy, specify the proxy settings on this page.

Item	Description
No proxy	Click this radio button to connect to the Internet without a proxy.
Auto-detect proxy settings	Click this radio button to enable using an auto-configuration URL to configure the web proxy settings. When this option is selected, the following controls become enabled:

Item	Description
Automatic proxy configuration URL	Select this check box to manually specify the location of the proxy settings file, in case IntelliJ IDEA does not find it automatically.
Clear passwords	Click this button to clear the passwords to the specified proxy.

Item	Description
Manual proxy configuration	Click this radio button to enable manual proxy configuration. When this option is selected, the following controls become enabled:

Item	Description
HTTP	Click this radio button if you want IntelliJ IDEA to use an HTTP proxy when accessing the Internet.
SOCKS	Click this radio button if you want IntelliJ IDEA to use the Socket Secure protocol when accessing the Internet.
Host name	Specify the proxy hostname or IP address.
Port number	Specify the proxy port number.
No proxy for	Specify here the patterns for the URLs or IP addresses, for which proxy should not be specified.
Proxy authentication	Select this check box if your proxy requires authentication.
Login	Specify the name of the user on whose behalf IntelliJ IDEA will connect to the proxy.
Password	Specify the password associated with the user name (login).
Remember password	Select this check box if you want IntelliJ IDEA to remember the password. Otherwise, you will be asked to provide the password every time IntelliJ IDEA connects to the proxy.

Updates

File | Settings | Appearance and Behavior | System Settings | Updates for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | System Settings | Updates for OS X

Ctrl+Alt+S

Use this page to:

- Enable automatic update of IntelliJ IDEA and specify to which kind of release you want it updated.
- Obtain information about the current IntelliJ IDEA version and availability of a newer version.

Item	Description
Check for updates for	Select this check box to enable the automatic update function, and select the desired update channel (for example, stable version). – Channel Early Access Program: this channel gets patch from the previous EAP/release version. This is not recommended for production development. More details about the Early Access Program, or EAP, are available at http://eap.jetbrains.com/ . – Channel Beta Releases or Public Previews: this channel includes release candidates (RC). – Channel Stable Releases: this channel includes all IntelliJ IDEA releases, for example, IntelliJ IDEA X.Y.Z Note that the list is only available for the stable versions. For the various EAPs, it is enforced to Early Access Program.
Use secure connection	– If this check box is selected, the secure connection protocol (HTTPS) is used. – If this check box is cleared, the HTTP protocol is used. Note that the HTTP protocol may be blocked due to security reasons. By default, the check box is selected.
Check Now	Click this button to check for updates immediately. Tip You can alternatively choose Help | Check for Updates (for Windows or *NIX) or IntelliJ IDEA | Check for Updates (for OS X) on the main menu.
View/edit ignored updates	Follow this link to show/change the builds which were ignored on IntelliJ IDEA update. These build numbers are included in the list of ignored updates and not suggested any more.

Usage Statistics

File | Settings | Appearance and Behavior | System Settings | Usage Statistics for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | System Settings | Usage Statistics for OS X

Ctrl+Alt+S

Use this page to share the statistics of your IntelliJ IDEA usage with JetBrains.

Item	Description
Allow to send usages statistics to JetBrains	Select this check box to allow JetBrains to collect your anonymous statistics.
Daily, Weekly or Monthly	Select one of these options to define how often your usage statistics should be sent to JetBrains.

File Colors

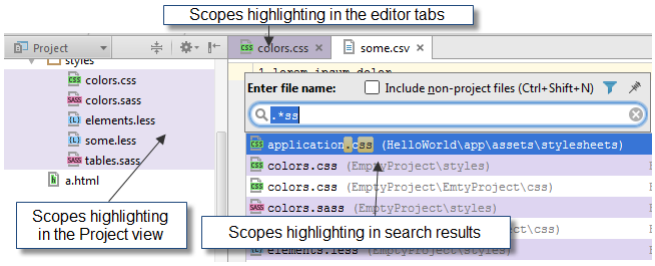
File | Settings | Appearance and Behavior | File Colors for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | File Colors for OS X

Ctrl+Alt+S

Use this page to set different background colors for distinguishing between project files , folders, and packages of specific scopes . The settings apply to the following UI elements:

- The headers of editor tabs.
- Navigation lists when one searches for files or classes by their names
- Project view



On this page:

- Common Options
- Local Colors
- Shared Colors

Common Options

Item	Description
Enable File Colors	Select this check box to apply background color settings to navigation lists .
Enable Colors in Editor Tabs	Select this check box to apply background color settings to the headers of editor tabs.
Use in Project View	Select this check box to apply background color settings to the Project view.
Manage Scopes	Click this button to open the Scopes dialog in which you can define custom scopes for various actions.

Local Colors

In this area, configure the color-scope associations to be applied locally.

Once defined, a color-scope association cannot be changed. To re-assign a color to a scope, remove the existing association and define a new one.

Item	Tooltip	Description
Scope		This read-only field shows the scope to apply the color setting to.
Color		This read-only field shows the color to be applied to the corresponding scope.
+	Add	Click this button to open the Add Color Label dialog in which you can configure a new color-scope association.
−	Remove	Click this button to remove the selected color-scope association.
↑ or ↓	Move up or Move down	Use these buttons to resort the color-scope associations and thus determine the order in which they are applied.
	Share	Click this button to have the selected scope-color association shared among the members of the team. The selected association will be accordingly moved to the list in the Shared Colors area.

Shared Colors

Use the controls in this area to configure the color-scope associations to be shared among all the members of the team.

Once defined, a color-scope association cannot be changed. To re-assign a color to a scope, remove the existing association and define a new one.

Item	Tooltip	Description
Scope		This read-only field shows the scope to apply the color setting to.
Color		This read-only field shows the color to be applied to the corresponding scope.
+	Add	Click this button to open the Add Color Label dialog in which you can configure a new color-scope association.
−	Remove	Click this button to remove the selected color-scope association.
↑ or ↓	Move up or Move down	Use these buttons to resort the color-scope associations and thus determine the order in which they are applied.
	Unshare	Click this button to have the selected scope-color association applied only locally. The selected association will be accordingly moved to the list in the Local Colors area.

Scopes

File | Settings | Appearance and Behavior | Scopes for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | Scopes for OS X

`Ctrl+Alt+S`

A scope is a set of files to which various operations apply. Using this dialog, you can define scopes for the various IntelliJ IDEA actions, for example, Find Usages , or Code Inspections .

In this section:

– Main toolbar
– Scope configuration controls
– Examples
– Scope toolbar

Main toolbar

Item	Tooltip	Description
	Add scope `Insert`	Click this button to add a new local or shared scope.
	Delete `Delete`	Click this button to delete the selected scope from the list.
	Copy configuration `Ctrl+D`	Click this button to create a copy of the selected scope.
	Save as	Click this button to have the selected local scope saved as shared or a selected shared scope as local.
	Move Up/Move Down	Use these buttons to move the scopes up and down in the list. If some file is included into several scopes, the order of the scopes becomes important: IntelliJ IDEA uses the color of the uppermost scope (shown in the Scopes settings page) to highlight such file. Of course, you can change the order of the scopes, and thus the resulted highlighting. .

Scope configuration controls

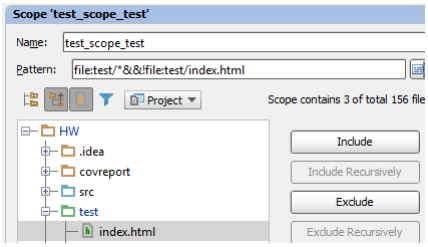
Item	Description
Name	In this text box, specify the scope name.
Pattern	In this text box, specify the pattern that defines the current scope. The following elements and structures can be used: 1. The `file:` modifier. The element is mandatory. 2. The `*` asterisk to denote any symbol in a file name or file extension. 3. Logical operators `AND (&&)`, `OR (||)` , and `NOT (!)` . For more information, see Scope Language Syntax Reference . Do one of the following: – Type or edit the pattern manually in the text field Pattern. – Click , or press `Shift+Enter` to type of edit in the Pattern dialog box. – Choose the desired files in the Project Tree View and use the buttons described below to make IntelliJ IDEA generate the corresponding pattern automatically. **Warning!** Storing empty or incorrect patterns is not allowed. In such cases, you will be prompted with the Syntax Error warning.
Include	Click this button to have the selected element included in the scope. The corresponding expression is automatically generated and added to the expression in the Pattern text box. **Tip** If the current element is a folder, the nested subfolders are ignored.
Include Recursively	Click this button to have the selected folder included in the scope, together with the nested subfolders. The corresponding expression is automatically generated and added to the expression in the Pattern text box.
Exclude	Click this button to have the selected element excluded from the scope. The corresponding expression is automatically added to the Pattern. If the current element is a folder, the nested subfolders are ignored.

Examples

– `file[MyMod]:src/main/java/com/example/my_package//*` - include in a project all the files from module "MyMod", located in the specified directory and all subdirectories.
– `src[MyMod]:com.example.my_package..*` - recursively include all classes in a package in the source directories of the module.
– `lib:com.company..*||com.company..*` - recursively include all classes in a package from both project and libraries.
– `test:com.company.*` - include all test classes in a package, but not in subpackages.
– `[MyMod]:com.company.util.*` - include all classes and test classes in the package of the specified module.
– `file:*.js||file:*.coffee` - include all JavaScript and CoffeeScript files.
– `file:*js&&!file:*.min.*` - include all JavaScript files except those that were generated through minification, which is indicated by the `min` extension.

Exclude Recursively	Click this button to have the selected folder excluded from the scope, together with the nested subfolders. The corresponding expression is automatically added to the Pattern field.

Scope toolbar

		ItemTooltipDescription
Project tree view		The tree view contains all the files available in your project. In the view, select the desired files to be included in the current scope and have the scope definition pattern generated automatically. The message on the toolbar shows the total number of available files and the number of files included in the scope. See also the color legend below. Use the toolbar buttons described below to change the view presentation. The way of presentation of the project tree view, the available controls and scopes language syntax depend on the selection in project vs package drop-down list.
	Flatten Packages	When the button is pressed, all the packages are displayed as a single-level tree view. This enables you to find a package somewhere deep within the project by its name without going through the entire tree hierarchy.
	Compact Empty Middle Packages	This option lets you specify how or whether empty packages are to be shown. (Empty packages are ones that contain nothing but other packages.)
	Group by Scopes Type	When this button is pressed, the items in the tree-view are grouped below three nodes: – Library Classes – Production Classes – Test Classes This button is only available for the Package presentation of scopes.
	Show Files	If this button is pressed, source files are displayed explicitly in the tree view. When the files are shown, they can be selected for exclusion/inclusion into a pattern.  If this button is not pressed, the files are hidden.
	Show Included Only	When the button is pressed, the tree shows only the elements that are included in the scope.
		Use this drop-down box to define how you want the project files to be displayed in the tree view. The available options are: – Project – Packages The project tree view presentation, scopes language syntax and the available toolbar buttons differ depending on the selection.

Legend of the project tree view

	ItemDescription
Green	Folders and files included in scope.
Black	Folders and files excluded from scope.
Blue	Folders that contain both excluded and included files and subfolders.

Notifications

File | Settings | Appearance and Behavior | Notifications for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | Notifications for OS X

Ctrl+Alt+S

Use this page to enable and disable notifications about certain events, change their presentation, and optionally enable their logging.

Item	Description
Display balloon notifications	Select this check box to enable event notifications for IntelliJ IDEA. (The notifications, generally, are shown in the balloons that appear on the screen when the corresponding events take place.)
Enable system notifications	Select this check box to allow showing system notification. Warning! This option is not available on platforms where system notifications are not supported (Windows and some Unixes).
Group	This column lists groups of events that you may be notified of and/or that may be logged.
Popup	If the Display balloon notification check box is selected, the settings in this column specify how the notifications for the corresponding group of events are shown. The available display options are: – Balloon: The balloons with the notification messages appear on the screen for a short period of time and then disappear automatically. The notifications are also shown in the Status bar, and added to the list of notifications. – Sticky balloon: The notification balloons stay on the screen unless you close them. – Tool window balloon: The notification balloons are shown only if an appropriate tool window is open. – No popup: The notifications for the corresponding group of events are not shown.
Log	If the check box for a group of events is selected, the corresponding events are logged and can be seen in the Event Log tool window.

Quick Lists

File | Settings | Appearance and Behavior | Quick Lists for Windows and Linux

IntelliJ IDEA | Preferences | Appearance and Behavior | Quick Lists for OS X

Ctrl+Alt+S

Use this page to configure quick lists. A Quick List is a pop-up menu of IntelliJ IDEA commands, configured by the user and associated with a keyboard or mouse shortcut. You can create as many quick lists, as necessary. Each command, included in a quick list, is identified by a sequential number. Numbering starts from the numerals (0 to 9), and then proceeds with the letters in alphabetical order.

Item	Shortcut	Description
+	Alt+Insert	Create a new Quick List.
−	Alt+Delete	Delete the selected Quick List.
Display name		Edit the name of the selected Quick List.
Description		Edit the description of the selected Quick List. (The description is optional.)
+	Alt+Insert	Use this button to add actions to the Quick List. Select the actions in the Add Actions to Quick List dialog that opens.
-----		Use this button to add a separator at the end of the Quick List. (Separators help you organize menu commands in logical groups.)
−	Alt+Delete	Remove selected actions from the Quick List.
↑	Alt+U	Use this button to move the selected item one line up in the list.
↓	Alt+D	Use this button to move the selected item one line down in the list.

Path Variables

File | Settings | Appearance & Behavior | Path Variables for Windows and Linux

IntelliJ IDEA | Preferences | Appearance & Behavior | Path Variables for OS X

`Ctrl+Alt+S`

On this page:

– Path Variables page . Use this page to configure path variables and the list of ignored variables .
– Add / Edit Variable dialog . Use this dialog to specify the name and value for a new or existing path variable.
– Example .

Path Variables page

Item	Description
Reset	Use this link to revert path variables and the list of ignored variables to their initial saved states.
Name	This field shows the name of a path variable (readonly).
Value	This field shows the value of a path variable (readonly).
+ (`Alt+Insert`)	Use this icon or shortcut to create a new path variable. (The Add Variable dialog will open.)
(`Enter`)	Use this icon or shortcut to edit the selected path variable. (The Edit Variable dialog will open.)
– (`Alt+Delete`)	Use this icon or shortcut to delete the selected path variable.
Ignored Variables	List the names of the variables that should be ignored . Use semicolons (`;`) to separate individual list items.

Add / Edit Variable dialog

Item	Description
Name	Specify the path variable name.
Value	Specify the path variable value. Use … (`Shift+Enter`) to select the necessary file or folder in the Select Path dialog .

Example

Consider storing a library on your disk. This library is attached to your project, and the path to this library is included in the `*.iml` file of your project. However, this path should not be absolute, since the other teammates may store same library in the different locations.

That's why it makes sense to create a dedicated path variable `PATH_TO_LIB` :

1. On the Settings dialog, click Path Variables .
2. Click +.
3. In the Add Variable dialog box, type the variable name `PATH_TO_LIB` , and its value that points to the library location on your disk.
4. Share the `*.iml` file on the version control.
5. The other developers should update their projects, and change the value of `PATH_TO_LIB` variable to point to the locations of their libraries.

Keymap

File | Settings | Keymap for Windows and Linux IntelliJ IDEA | Preferences | Keymap for OS X

Use this page to create, edit, and remove custom keymaps for specific environments, and change shortcuts associated with actions.

Note that default keymaps are not editable. To re-configure shortcut associations, create a child keymap based on the desired default one and edit it as required.

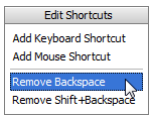
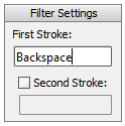
On the other hand, as soon as you try to change a keyboard shortcut associated with an action in one of the default keymaps, a copy of the corresponding keymap is automatically created.

– Keymap Management Buttons
– Keymap Toolbar
– Actions

Keymap Management Buttons

Item	Description
Keymaps	From this drop-down list, select the desired keymap.
Copy	Click this button to create a child keymap on the basis of the keymap selected in the Keymaps drop-down list.
Reset	Click this button to abandon all the changes made to a custom keymap and restore the configuration of the parent keymap.
Delete	Click this button to remove the selected custom keymap from the list.
<language> layout support	This check box appears when a non-English keyboard layout has been detected. When you type any character in the editor, the keyboard layout is recognized and a notification appears. **Warning!** This option is available for Windows and Linux only. If you don't see this option, it means either you are using OS X or IntelliJ IDEA didn't detect any of the supported input locales.
Based on keymap	This read-only field shows the name of the parent keymap.

Keymap Toolbar

Item	Tooltip and shortcut	Description
	Expand All `Ctrl+NumPad Plus`	Click this button to expand all nodes in the content pane of actions.
	Collapse All `Ctrl+NumPad -`	Click this button to collapse all nodes in the content pane of actions.
	Edit Shortcut `Enter`	Click this button to change shortcuts for the selected action. It is possible to remove existing shortcuts, and add new ones. Choose the desired change from the drop-down menu:  – Select the option Add Keyboard Shortcut to open the Enter Keyboard Shortcut dialog box, where you can specify the combination of keystrokes to be assigned to the selected action in the current keymap. – Select the option Add Mouse Shortcut to open the Enter Mouse Shortcut dialog box, where you can specify the combination of mouse clicks and buttons to be assigned to the selected action in the current keymap. – Select the options Remove <shortcut> to delete the selected shortcut from the selected action. These commands are duplicated on the context menus of the actions in the Actions content pane.
		Use this text box to search through the content pane of actions. As you type a search string, the actions that match the search pattern are displayed. The previously used search patterns are stored in the search history list. To add the search string to the history list, press `Enter`. Click to reveal the history list of the previous searches. Click to clear the current search pattern from the text box.
	Find Action by Shortcut	Click this button to open the Filter Settings dialog box for filtering out the desired actions by keystrokes. Refer to the section Configuring Keyboard Shortcuts to learn how to specify keyboard shortcuts.  The actions with shortcuts that match the specified criteria are shown in the content pane of actions.
	Clear Filtering	Click this button to restore the initial list of actions in the content pane.

Actions

Item	Description
All Actions	This content pane shows all actions currently available in IntelliJ IDEA. The actions are grouped below nodes according to the areas of their use.

- Note that default keymaps are not editable. As soon as you try to change a keyboard shortcut associated with an action, a copy of the corresponding keymap is automatically created.
- If some of the actions have no mapped keyboard shortcuts, they still can be invoked by Go to Action .

Shortcuts	This read-only field shows the list of shortcuts associated with the selected action in the current keymap. Tip The shortcuts are represented depending on the platform. However, some keys are missing on certain keyboard layouts. For example, PageUp / PageDown keys are not available on notebooks. That's why one should use Fn plus arrow keys.
Context menu of an action	
Add Keyboard Shortcut	Choose this command on the context menu of an action to open the Enter Keyboard Shortcut dialog box, where you can specify the combination of keystrokes to be assigned to the selected action in the current keymap.
Add Mouse Shortcut	Choose this command on the context menu of an action to open the Enter Mouse Shortcut dialog box, where you can specify the combination of mouse clicks and buttons to be assigned to the selected action in the current keymap.
Add abbreviation	Choose this command to add an abbreviation that can be used in Search Everywhere .
Remove <shortcut>/<abbreviation>	Choose this command on the context menu of an action to delete the selected shortcut or abbreviation.

Enter Mouse Shortcut Dialog

The dialog box opens when you select an action and click the Add Mouse Shortcut button. Use this dialog box to bind the selected action with a new mouse shortcut, which may be a single or a double clicking one of the mouse buttons or the wheel button.

The resulting mouse shortcut is marked with the icon in the Shortcuts list.

Item	Description
Click Count	In this area, specify the type of mouse click to be assigned to the selected action. The available options are: – Single Click – Double Click
Click Pad	Click the desired mouse button anywhere in this area. Tip The number of clicks in this area does not affect the shortcut configuration. No matter how many times you click a button, the click type chosen in the Click Count area will be assigned.
Shortcut Preview	This read-only field shows the newly defined shortcut.
Conflicts	This read-only field shows messages about conflicts that arise if a suggested mouse shortcut is already in use. Warning! You can ignore a conflict and assign a shortcut to several actions. However it is strongly recommended that you avoid binding two actions with the same shortcut, because the priority of these actions is not defined.

Enter Keyboard Shortcut Dialog

The dialog box opens when you select an action and click the Add Keyboard Shortcut button. Use this dialog box to bind the selected action with a new keyboard shortcut, which may consist of one or two keystrokes. The resulting keyboard shortcut is marked with the icon in the Shortcuts list.

Warning! Use your mouse pointer to click buttons in the dialog box. Any key stroke is interpreted as a shortcut!

Item	Description
First Stroke	Use this text box to define the primary shortcut by pressing keyboard keys and key combinations.
Enable	Select this check box to allow an optional second shortcut.
Second Stroke	Use this text box to define an optional shortcut by pressing keyboard keys and key combinations. This field is available after the Enable check box is selected.
Shortcut Preview	In this read-only field, view the newly defined shortcuts.
Conflicts	This read-only field shows messages about conflicts that arise if a suggested keystroke is already in use. Warning! You can ignore a conflict and assign a shortcut to several actions. However it is strongly recommended that you avoid binding two actions with the same shortcut, because the priority of these actions is not defined.

Editor

File | Settings | Editor for Windows and Linux

IntelliJ IDEA | Preferences | Editor for OS X

`Ctrl+Alt+S`

When you select the Editor category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- General
- Colors and Fonts
- Code Style
- Inspections
- File and Code Templates
- File Encodings
- Live Templates
- File Types
- Copyright
- Emmet
- GUI Designer
- Images
- Intentions
- Language Injections
- Spelling
- TextMate Bundles
- TODO

General

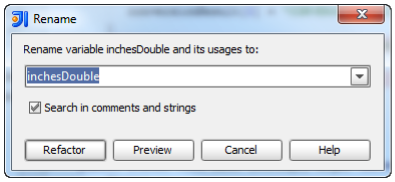
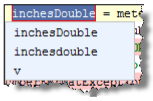
File | Settings | Editor | General for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General for OS X

Ctrl+Alt+S

Use the General page of the Settings dialog to configure the editor behaviour and customize its view.

Item	Description
Mouse	
Honor "CamelHumps" word settings when selecting using double click	Select this check box to have IntelliJ IDEA invoke the CamelHumps selection when words are selected by double-clicking. This feature works only if the Use 'CamelHumps' words option is enabled.
Change font size (Zoom) with Ctrl/Command+Mouse Wheel	If this check box is selected, a particular editor font size can be changed by rolling the mouse wheel while holding the Ctrl/⌘ key. This check box also affects font size in quick documentation lookup . If this option is unchecked, rolling the mouse wheel while holding the Ctrl/⌘ key scrolls the editor.
Enable Drag'n'Drop functionality in editor	If this check box is selected, you can drag-n-drop code fragments in the editor. Refer to Using Drag-and-Drop in the Editor .
Soft Wraps	
Use soft wraps in editor	If this check box is selected, soft wraps (or word wraps) are used in the editor. The horizontal scroll bar is not normally shown when this option is enabled. However, in certain cases, when a line cannot be "soft-wrapped", the horizontal scroll bar still appears (for example, if a line consists of a single string that is wider than the visible area.)
Use original line's indent for wrapped parts	Select this check box to use custom indentation for soft wraps on resizing the editor or console. Specify the indent value in the Additional shift text field on the right.
Show soft wrap indicators for current line only	If this check box is selected, the soft wrap characters will be shown in the active logical line only. Otherwise, soft wraps characters will be shown at the end of each line, and at the beginning of each next line.
Virtual Space	
Allow placement of caret after end of line	If this check box is cleared, the caret never rests after the last symbol in a line.
Allow placement of caret inside tabs	Select this check box to allow placing the caret inside tab characters. The reason is that each tab character shows in the editor as a set of 'virtual' space characters.
Show virtual space at file bottom	If this check box is selected, the currently edited line (even if it is the final line) can be scrolled to the top of the screen. IntelliJ IDEA adds the necessary amount of virtual lines.
Other	
Strip trailing spaces on Save	From this drop-down list, select the mode in which IntelliJ IDEA will handle trailing spaces in the end of lines on file saving: – Modified lines - Strips trailing spaces only in the end of modified lines. – All - Strips trailing spaces in all lines. – None - Does not strip trailing spaces.
Ensure line feed at file end on Save	Select this check box to have IntelliJ IDEA automatically add an empty line in the end of a file during the save procedure.
Show quick documentation on mouse move	Select this check box to show quick documentation for the symbol at caret. The quick documentation pop-up window appears after the specified delay.
Highlight modified lines in gutter	Select this check box if you want added/modified lines to be highlighted with a color stripe in the left editor gutter.
Different color for lines with whitespace-only modifications	This option only becomes available if the Highlight modified lines in gutter option is enabled. Select this check box if you want lines where only whitespaces were added/removed to be highlighted with a different color from lines with more significant modifications.
Highlight on Caret Movement	
Highlight matched brace	Select this check box to have IntelliJ IDEA highlight pairs of opening/closing braces when you position the caret right before the opening or right after the closing one. It also works for HTML and XML tags.
Highlight current scope	Select this check box to have IntelliJ IDEA highlight the available scope for the code typed in the current caret location.
Highlight usages of element at caret	Select this check box to have IntelliJ IDEA highlight all usages of the element at which the caret is currently positioned.
Formatting	
Show notification after reformat code action	Select this check box to show a notification with changes in your code and a shortcut to the Reformat Code dialog every time you try to reformat the code. Otherwise, IntelliJ IDEA will reformat code silently.
Show notification after optimize imports action	Select this check box to show notification with changes in your code. Otherwise, IntelliJ IDEA will optimize imports silently.
Scrolling	
Smooth scrolling	Select this check box to enable smooth scrolling in the editor.
Prefer scrolling editor canvas to keep caret line centered	Click this option to choose scrolling editor canvas and keeping the caret in place. Keeping the caret in place and scrolling the editor canvas can be helpful in course of debugging session . As you step through the lines of code, the

	editor canvas scrolls, while the line at caret is always in the center of the screen.
Prefer moving caret line to minimize editor scrolling	Click this option to choose moving the caret. When you step through the lines of code during the debugging session , the caret moves down, and the editor canvas doesn't scroll until the caret line reaches the bottom of the screen.
Refactorings	
Enable in-place mode	Select or clear this check box to enable or disable in-place refactorings for Java. The in-place in connection with the refactorings means specifying all or most of the information necessary for the refactoring by typing, right in the editor. All the affected code fragments are highlighted and change as you type. If appropriate, additional refactoring options are selected in corresponding option boxes. The in-place refactoring mode is available for the following refactorings: – Extract Constant – Extract Field – Extract Parameter – Extract Variable – Rename If this check box is not selected, the refactoring settings for all of the refactorings are specified in the corresponding dialogs.
Preselect old name	If this check box is selected, the old name of a symbol is selected when the Rename refactoring is invoked for that symbol.   If check box is not selected, the symbol being renamed is not selected.
Show inline dialog for local variables	Select this check box if you want to display a confirmation dialog for the "Inline local variable" refactoring.
Limits	
Maximum number of contents to keep in clipboard	In this text box, specify how many code blocks can be kept in clipboard.
Recent file limit	In this text box, specify how many file names can be included in the list of recent files.
Rich-text copy	
Copy as rich text by default	Select this check box to copy a rich text from the editor to any other editor that recognizes RTF. Note that you can override this option if you select Copy as Plain Text from the context menu in your editor and vice versa, using the Copy as Rich Text option from the context menu overrides the unselected check box in the editor settings.
Color scheme	Use this drop-down list to select a color scheme for the text copy. You can select from the following options: – Default – Active scheme – Darcula
Error highlighting	
Error stripe mark min height (pixels)	In this text box, specify the minimum size of the error and warning stripes.
Autoreparse delay (ms)	In this text box, specify the time period after which IntelliJ IDEA starts reparsing the entered text.
'Next Error' action goes to high priority problems only	Select this check box to have IntelliJ IDEA pass through the highest priority problems only (for example, errors), when executing Navigate | Next/Previous Highlighted Error command (F2 / Shift+F2). Clear this check box to have IntelliJ IDEA pass through all the existing problems (for example, errors and warnings) sequentially.
Suppress with @SuppressWarnings (for 5.0 only)	Select this check box to have `@SuppressWarnings` implemented as an annotation. Clear this check box to have `@SuppressWarnings` implemented as a JavaDoc comment.

Auto Import

File | Settings | Editor | General | Auto Import for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General | Auto Import for OS X

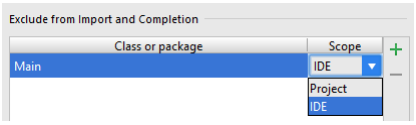
Ctrl+Alt+S

- XML
- Java
- TypeScript
- JSP
- Scala
- PHP

XML

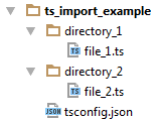
Show import pop-up	Automatically display an import pop-up dialog box when typing the name of an unbound namespace.

Java

Insert imports on paste	Use this drop-down list to define how IntelliJ IDEA will insert imports for pasted blocks of code, if they contain references to classes that are not imported into the target class. The available options are: – All - select this option to have IntelliJ IDEA automatically add import statements for all classes that are found in the pasted block of code and are not imported in the current class yet. – Ask - if this option is selected, when pasting code blocks, IntelliJ IDEA will open a dialog box, where you can choose the desired imports. – None - select this option to suppress import. Tip If you skip an import suggested in the Ask mode or choose the None mode, the non-imported classes will be red-highlighted and an import pop-up window will appear to help you create import statements using the Alt+Enter keyboard shortcut.
Show import pop-up	Automatically display import pop-up dialog box when typing the name of a symbol that lacks import statement.
Optimize imports on the fly	Select this check box to have have the Optimize Imports operation automatically performed for your files.
Add unambiguous imports on the fly	Select this check box to have IntelliJ IDEA automatically add imports that can be added without user intervention.
Exclude from Import and Completion	In this area, create a list of packages and classes that should not be automatically included in the import statements. Note that you can exclude packages and classes both on the project level and on the IDE level. This can be done in the cells of the Scope column:  Use: – + (Alt+Insert) to enter the name of the class/package to be excluded from import and completion. – – (Alt+Delete) to remove the selected item from the list.

TypeScript

Show import pop-up	Automatically display import pop-up dialog box when typing the name of a symbol that lacks import statement.
Merge imports for symbols from the same module	

Use paths relative to tsconfig.json	When this check box is selected, IntelliJ IDEA calculates import paths using the `tsconfig.json` file as the root. When this check box is cleared, IntelliJ IDEA calculates import paths relative to the project root.
	For example, if your project is structured as follows:
	 
	
	With the check box selected, IntelliJ IDEA generates the following import statement:
	`import {ClassName} from 'directory_2/file_2'`
	
	If the check box is cleared, the following import statement is generated:
	`import {ClassName} from '../directory_2/file_2'`

JSP

Add unambiguous imports on-the-fly	Select this check box to have IntelliJ IDEA automatically add imports that can be added without user intervention.

Scala

Note This table is only available when Scala plugin is downloaded and installed. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins.

Insert imports on page	Use this drop-down list to define how IntelliJ IDEA will insert imports for pasted blocks of code, if they contain references to classes that are not imported into the target class. The available options are: – All - select this option to have IntelliJ IDEA automatically add import statements for all classes that are found in the pasted block of code and are not imported in the current class yet. – Ask - if this option is selected, when pasting code blocks, IntelliJ IDEA will open a dialog box, where you can choose the desired imports. – None - select this option to suppress import. Tip If you skip an import suggested in the Ask mode or choose the None mode, the non-imported classes will be red-highlighted and an import pop-up window will appear to help you create import statements using the Alt+Enter keyboard shortcut.
To disable import import pop-up, use Java settings	Refer to the same options in the settings for Java.
Add unambiguous imports on the fly	Select this check box to have IntelliJ IDEA automatically add imports that can be added without user intervention.
Optimize imports on the fly	Select this check box to have have the Optimize Imports operation automatically performed for your files.

PHP

Note This table is only available when PHP plugin is downloaded and installed. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins.

Enable auto-import in file scope	Select this check box to have IntelliJ IDEA automatically import PHP namespaces, add use statements, and complete short class names on the fly, when you are typing in a class or file that does not belong to any specific namespace. By default, the check box is cleared.
Enable auto-import in namespace scope	Select this check box to have IntelliJ IDEA automatically import PHP namespaces, add use statements, and complete short class names on the fly when you are typing in a class or file that belongs to a certain namespace. The check box is selected by default.

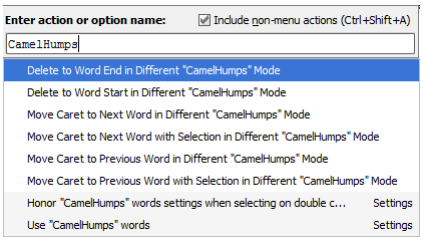
Smart Keys

File | Settings | Editor | General | Smart Keys for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General | Smart Keys for OS X

`Ctrl+Alt+S`

Use this page to enable or disable specific smart keys and to define which actions you want to be invoked automatically.

Item	Description
Home	When this check box is selected, on pressing `Home`, the caret is positioned at the first non-space character of the current line. Pressing `Home` subsequently moves the caret from the *Smart Home position* to the first column and back.
End (on blank line)	When this check box is selected, on pressing `End` in an empty line, the caret is positioned with the indent, which IntelliJ IDEA assumes to be reasonable in the current code point (indentation is based on the current Code Style Settings).
Insert pair bracket	Select this check box to have IntelliJ IDEA automatically add a closing round or square bracket for each typed opening round or square bracket, respectively.
Insert pair quote	Select this check box to have IntelliJ IDEA automatically add a closing single or double quote for each typed opening single or double quote, respectively.
Reformat block on typing "}"	If this check box is selected, then, on typing the closing curly brace, the enclosed code block is reformatted automatically, if the formatting of this code block does not match the selected code style.
Use 'CamelHumps' words	Select this check box to have IntelliJ IDEA discern separate words within CamelHump names. Words within a name should start with a capital letter or an underscore. This option impacts some editor actions, for example: – Caret Move (`Ctrl+Right` / `Ctrl+Left`) – Caret Move with Selection (`Ctrl+Shift+Right` / `Ctrl+Shift+Left`) – Select Word at Caret (`Ctrl+W`) – Delete to Word Start/End (`Ctrl+Backspace` and `Ctrl+Delete` respectively) – Double-clicking The following editor actions work in a mode opposite to the one established by the Use 'CamelHumps' words setting: – Move Caret to Previous Word in Different "CamelHumps" mode – Move Caret to Previous Word with Selection in Different "CamelHumps" mode – Move Caret to Next Word in Different "CamelHumps" mode – Move Caret to Next Word with Selection in Different "CamelHumps" mode – Delete to Word End in Different "CamelHumps" mode – Delete to Word Start in Different "CamelHumps" mode For example, If the check box is selected, the action Move Caret to Next Word in Different "CamelHumps" mode moves the caret to the end of word; if the check box is not selected, then the caret moves to the next CamelHump. These actions have no default keyboard shortcuts, and are not included in the menus. Thus, they can be invoked by pressing `Ctrl+Shift+A`:  (Refer to the section Navigating to Action) You can bind them with the shortcuts of your choice as described in the section Configuring Keyboard Shortcuts .
Surround selection on typing quote or brace	If this check box is selected, the selected text on typing a quote, double-quote or brace, will be surrounded with these characters. If this check box is not selected, then the typed quotes, double-quotes or braces will replace the selection.
Add multiple carets on double `Ctrl` / `⌥` with arrow keys	If this check box is selected, then: – pressing `Ctrl` (for Windows or *NIX) or `⌥` (for Mac OS) twice plus up/down arrow keys leads to creating multiple carets. – pressing `Ctrl` (for Windows or *NIX) or `⌥` (for Mac OS) twice plus left/right arrow keys or Home/End leads to creating a selection.
Enter	Use this area to define the actions to be invoked by pressing `Enter`. – Smart Indent - select this check box to have IntelliJ IDEA add a new line and position the caret in it, with the indent that IntelliJ IDEA assumes to be reasonable in the current point of code (indentation is based on the current Code Style Settings). If the check box is cleared, upon pressing `Enter` in a blank line, IntelliJ IDEA adds a new line and positions the caret at the current non-space character column. – Insert pair '}' - select this check box to have IntelliJ IDEA automatically position a closing brace `}` at the proper column when `Enter` is pressed in an empty line. In this case IntelliJ IDEA seeks backward for the nearest unclosed opening brace `{` and places the closing one at the corresponding indentation level. – Insert documentation comment stub - this check box defines the behavior on pressing `Enter` after an opening documentation tag. – If this check box is selected, IntelliJ IDEA generates a documentation comment stub. For the method comments, this stub contains the required tags (`@param` tags for each method parameter, `@return` , or `@throws`). Refer to Creating Documentation Comments and Creating JSDoc Comments for details. – If this check box is not selected, only the closing tag is generated. Warning! Note that this check box refers to JavaScript, Java and the other languages that have special beginning of documentation comments.
Backspace	Use this drop-down list to define the actions to be invoked by pressing `Backspace` key. The available options are:

- Disabled - pressing `Backspace` returns the caret by one position at a time.
- To nearest indent position
- To proper indentation

Reformat on paste

Use this drop-down list to specify how to place pasted code blocks. The available options are:
- None - The pasted code is inserted at the caret location as plain text without any reformatting or indenting.
- Indent Block - The pasted code block is positioned at the proper indentation level, according to the current Code Style Settings , but its inner structure is not changed.
- Indent Each Line- Each line of the pasted code block is positioned at the proper indentation level, according to the current Code Style Settings.
- Reformat Block- The pasted code block is reformatted according to the current Code Style Settings.

Tip This feature is applicable to lines that contain the trailing line feed characters.

XML/HTML

In this area, define the actions to be invoked automatically when editing XML or HTML code.
- Insert closing tag on tag completion: select this check box to have IntelliJ IDEA automatically insert a closing XML or HTML tag upon entering the corresponding opening one.
- Insert required attributes on tag completion: select this check box to have IntelliJ IDEA display a template with all mandatory attributes of the typed tag.
- Insert required subtags on tag completion: select this check box to have IntelliJ IDEA display a template with all mandatory subtags.
- Start attribute on tag completion: select this check box to have IntelliJ IDEA display a template with the first mandatory attribute of the typed tag.
- Add quotes for attribute value on typing '=': select this check box to have IntelliJ IDEA automatically add quotes for the value of the attribute that you are currently typing.
- Auto-close tag on typing '</': select this check box to automatically add a closing tag after entering </. Clear this check box to turn off such auto-completion.
- Simultaneous <tag></tag> editing:
  - When this check box is selected and you edit an opening tag the corresponding closing tag is automatically changed accordingly.
  - If this check box is cleared, editing the opening tag does not affect the closing tag which remains unchanged. As a result, the opening and closing tags do not match and the entire construct is underlined as erroneous.

  This check box controls the behaviour of IntelliJ IDEA in the following contexts:
  - HTML files
  - HTML injections within JavaScript code
  - HTML with templates Handlebars/Mustache templates
  - Handlebars/Mustache template files with the extension `.hbs`
  - XML, XHTML files
  - DTD files
  - JSX files
  - JSP files
  - HTML injections in PHP files

CSS

In this area, define the selection of CSS identifiers/classes:
- Select whole CSS identifier on double-click: If this check box is selected, double-click on a CSS identifier or class name selects the entire name up to the prefix:

  ```
  .ic-arrow-left
  ```

  If this check box is not selected, double-click on a CSS identifier or class name selects a portion of a name up to the nearest hyphens:

  ```
  .ic-arrow-left
  ```

AngularJS

Use this area to define the behavior of AngularJS:
- Auto-insert white space in the interpolation: If this check box is selected, a white space is automatically inserted between the braces: `{{ }}` .

  If this check box is not selected, the white space is not inserted: `{{}}` .

Javadoc

In this area, define the behavior of the closing tags in Javadoc.
- Automatically insert closing tag - select this check box to make IntelliJ IDEA insert closing tag automatically, after typing `>` . So doing, the caret rests inside the tag. For example, if you type `<b>` , the closing tag `</b>` will be generated automatically.

Insert pair '%>' on Enter in JSP

Select this check box to have IntelliJ IDEA automatically position the opening angle bracket `<` at the proper column when entered in an empty line in JSP code. In this case IntelliJ IDEA seeks backward for the nearest unclosed angle bracket and places a closing one `>` at the corresponding indentation level.

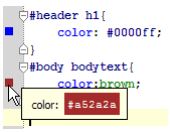
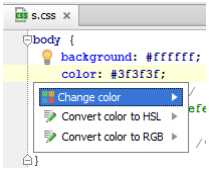
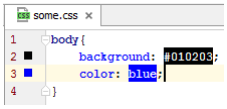
Appearance

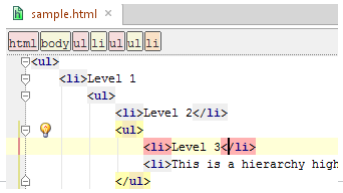
File | Settings | Editor | General | Appearance for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General | Appearance for OS X

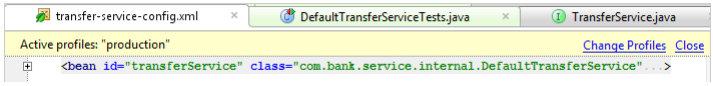
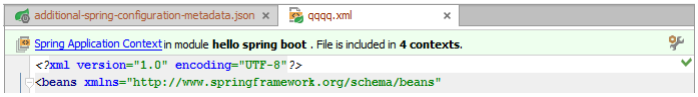
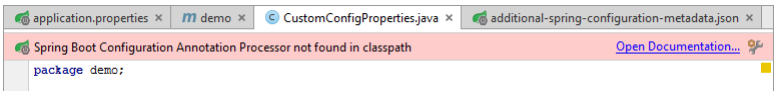
Ctrl+Alt+S

Use this page to customize the appearance of the Editor.

Item	Description
Caret blinking (ms)	Select this check box to make the caret blink with the specified period (in milliseconds).
Use block caret	Select this check box to have the block caret applied in the Insert mode and the usual caret applied in the Overwrite mode. Clear this check box to have the usual caret applied in the Insert mode and the block caret applied in the Overwrite mode.
Show right margin (configured in Code Style options)	Select this check box to have a thin vertical line at the right margin of the editor displayed. Refer to the description of the Code Style settings .
Show line numbers	Select this check box to have line numbering shown in the left gutter area.
Show method separators	Select this check box to have thin lines displayed in classes to separate methods from each other and to separate methods from field declarations.
Show whitespaces	Select this check box to have IntelliJ IDEA display white spaces or tabs (depending on the Code Style settings). You can select the following options: – Leader - select this check box to add white spaces before your code line. – Inner - select this check box to display white spaces inside the line of your code. – Trailing - select this check box to display white spaces after the code line.
Show vertical indent guides	Select this check box to have IntelliJ IDEA display vertical lines in the editor to indicate positions of indents and thus facilitate typing, manual formatting, reading, and maintaining code.
Show icons preview in gutter for small icons (Java)	If this check box is selected, preview of small icons (up to 16x16 pixels) are shown in the left gutter:
Show code lens on scrollbar hover	Select this check box to enable lens mode .
Show breadcrumbs	Select this check box to show a breadcrumb trail on top of the editor tab for an HTML or an XML file. Reopen the editor for the changes to take effect. (XML) (HTML)
Show CSS color preview icon in gutter	Select this check box to show color preview icons for the color values. See Changing Color Values in Style Sheets .  If this check box is not selected, it is still possible to invoke the color picker and change color values, by choosing the Change color intention action. 
Show CSS color preview as background	If this check box is selected, the background of the color value shows the color preview: 
Enable XML/HTML tag tree highlighting	Select this check box to show the hierarchy of tags highlighted with different colors. If this option is enabled, you can define the following options: – Levels to highlight: specify the depth of hierarchy to be highlighted. – Opacity: specify brightness of highlighting



Highlighting is activated when there is more than one tag with the same name in the hierarchy.

Show Spring Profiles panel	If this check box is selected, Spring configuration files show the active profile names on top of the editor. This panel also provides the Change Profiles action.  Clicking this action opens up the Change Active Spring Profiles dialog where you can choose a scope to which you want to apply the selected profiles. If there are no profiles in a file set, the panel is not displayed.
Show Spring Multiple Contexts panel	If this option is selected, a panel notifying you that a file belongs to different filesets is displayed.  Clicking the Spring Application Context link allows you to switch between different contexts.
Show Spring Boot meta-data panel	If this option is selected, a panel notifying you that Spring Boot Configuration Annotation Processor is not found in classpath will be displayed when you create a top-level class annotated with `@ConfigurationProperties` or the `additional-spring-configuration-metadata.json` file. 

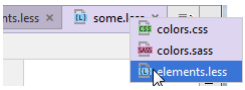
Editor Tabs

File | Settings | Editor | General | Editor Tabs for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General | Editor Tabs for OS X

`Ctrl+Alt+S`

Use this page to configure the appearance of the editor tabs and tab headers, specify their positioning on the screen, and define the tab closing policy.

Item	Description
Tab Appearance	
Placement	Use this drop-down list to define the location of the editor tab. The available options are: – Top - the default setting. – Bottom – Right – Left – None - select this option to have single editor without any tabs displayed. Refer to the section Changing Placement of the Editor Tab Headers for details.
Show tabs in single row	Select this check box to have headers of currently opened editor tabs displayed in a single row. As a result, some tab headers may become invisible. To cope with this problem, use the command Window | Editor Tabs | Show All Tabs. Refer to the section Navigating Between Editor Tabs . If this check box is selected, the sorting in alphabetical order mode becomes available for the top and bottom placement of the editor tabs. If this check box is not selected, headers of all the currently opened tabs are displayed, possibly, in several rows.
Hide tabs if there is no space	If this check box is selected, IntelliJ IDEA shows as many tabs as fits into the current IntelliJ IDEA frame; the rest of the tabs are hidden under the drop-down:  If this check box is not selected, all the editor tabs are shown; so doing, each tab's size reduces: This check box becomes enabled when Show tabs in single row check box is selected.
Hide file extensions in editor tabs	Select this check box to have only file names displayed in editor tab headers.
Show directory in editor tabs for non-unique file names	If this check box is selected, the editor tabs will show the file name together with the parent directory name; if this check box is not selected, only the file name will be included in the editor tab.
Show "close" button on editor tabs	Select this check box to have the Close Active Editor button displayed in editor tab headers.
Mark modified tabs with asterisk	If this check box is selected, changed but yet unsaved files have an asterisk on their editor tabs.
Show tabs tooltips	If this check box is selected, a tooltip with the complete path to a file displays on hovering the mouse pointer over a tab. If this check box is not selected, a tooltip is not shown.
Tab Closing Policy	
Tab limit	In this text box, specify the maximum number of the editor tabs to display.
Navigation from non-modified tab will reuse it	Use this option to specify the IntelliJ IDEA behaviour on `Ctrl+Click` (see section Viewing Definition for details). This option allows you to avoid cluttering of the editor space. If this check box is selected, then, if a file in an editor tab has not been modified and the users has navigated from this file, the target file opens in the same tab. If a file has been modified, then the target file opens in a new tab. If this check box is not selected, the target file always opens in a new tab. Note that a file is considered modified if its VCS status has changed .
When number of opened editors exceeds tab limit	In this area, specify which editor tab should be closed when the tab limit is reached and the user attempts to open a new file. The available options are: – Close non-modified files first - if this option is selected, IntelliJ IDEA examines the tabs in the order they were opened and closes the first tab with content that has not been modified. – Close less frequently used files - if this option is selected, IntelliJ IDEA closes the tab with the less frequently modified content.
When closing active editor	In this area, specify which editor tab to activate when closing the currently active tab. The available options are: – Activate left neighbouring tab - if this option is selected, IntelliJ IDEA activates the closest tab to the left from the tab being closed. – Activate right neighbouring tab - if this option is selected, IntelliJ IDEA activates the closest tab to the right from the tab being closed. – Activate most recently opened tab - if this option is selected, IntelliJ IDEA activates the tab with the file which was opened last.

Gutter Icons

File | Settings | Editor | General | Gutter Icons for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General | Gutter Icons for OS X

Ctrl+Alt+S

Use this page to hide or show the icons in the gutter area that invoke actions related to the basic, IntelliJ IDEA-wide features or to framework- and technology-specific features for all the newly created editors.

The right-hand pane shows all the gutter icons available in IntelliJ IDEA. The basic, IntelliJ IDEA-wide features, such as, (Run), are displayed at the top of the list under the Common title. Other features are grouped by the frameworks and technologies to which they are related , for example, Groovy, Spring Support, etc .

Note that a group of technology-related features is displayed only if the corresponding plugin is installed and activated, see Enabling and Disabling Plugins and Installing, Updating and Uninstalling Repository Plugins for details.

- To have an icon displayed in the gutter area, find the icon or the corresponding action in the list and then select the check box next to it.
- To have an icon hidden, clear the check box next to it.

Code Folding

File | Settings | Editor | General | Code Folding for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General | Code Folding for OS X

Ctrl+Alt+S

Use this page to specify your code folding preferences.

Item	Description
Show code folding outline	Select this check box if you want the code folding toggles (, and) to be shown in the editor. Clear the check box to hide the toggles.
Collapse by default	Select the code fragments which should be folded by default, that is, when a file is first opened in the editor.

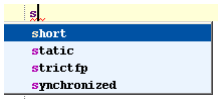
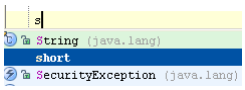
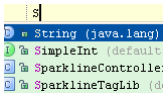
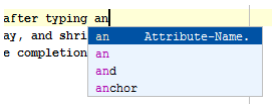
Code Completion

File | Settings | Editor | General | Code Completion for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General | Code Completion for OS X

Ctrl+Alt+S

Use this page to configure the code completion , and parameter information settings.

Item	Description
Code Completion	
Case sensitive completion	From this drop-down list, select the degree to which you want IntelliJ IDEA to take into consideration the case sensitivity when suggesting matches for code completion. The available options are: – All: The lookup list includes only those items that match the case of all typed letters. This option is most restrictive.  – None: The lookup list includes all matches regardless of their case.  – First letter: The lookup list includes only the items with the first letter matching. 
Auto-insert when only one choice on:	When the check boxes in this section are selected, IntelliJ IDEA doesn't show a lookup list for the corresponding completion type in cases when only one variant of code completion is available, and completes code automatically.
Sort lookup items lexicographically	If this check box is selected, the entries in the suggestion list will be sorted according to their lexical order. If this check box is not selected, the entries in the suggestion list will be sorted by relevance. Note that the check box defines the default behavior. You can change it any time by clicking the A or a icons in the suggestion list. Refer to the section Using Suggestion List for details.
Autopopup code completion	Select this check box, if you want suggestion list to appear after typing anything.  If the check box is not selected, IntelliJ IDEA will not suggest code completion automatically.
Insert selected variant by typing dot, space, etc.	If this check box is selected, code is completed by pressing certain character: comma, colon, semicolon, opening parentheses of the various kinds, equality sign, asterisk. This option is turned off by default.
Autopopup documentation (ms) For explicitly invoked completion	Select this check box to have IntelliJ IDEA automatically show a pop-up window with the documentation for the class, method, or field currently highlighted in the lookup list. In the text field to the right, specify the delay (in milliseconds), after which the pop-up window should appear. If this check box is not selected, use Ctrl+Q to show quick documentation for the element at caret. Quick documentation window will automatically pop up with the specified delay in those cases only, when code completion has been invoked explicitly. For the automatic code completion list, documentation window will only show up on pressing Ctrl+Q
Parameter Info	
Autopopup in (ms)	Select this check box to have IntelliJ IDEA automatically show a pop-up window with all available method signatures, when an opening bracket is typed in the editor, or a method is selected from the lookup list. In the text field to the right, specify the delay (in milliseconds) after when the pop-up window should appear. If this check box is not selected, use Ctrl+P to show the parameter info.
Show full signatures	If this check box is selected, the parameter info displays full signatures, including the method name and returned type.

Postfix Completion

File | Settings | Editor | General - Postfix Completion for Windows and Linux

IntelliJ IDEA | Preferences | Editor | General - Postfix Completion for OS X

Ctrl+Alt+S

– Overview

– Controls

Overview

Postfix code completion lets you transform an already typed expression to another one based on the postfix you type after a dot, the type of the expression, and its context. This transformation is performed by expanding the postfix-specific predefined template. For example, the `.if` postfix applied to a boolean expression wraps it with an `if` statement. See more at: Postfix Code Completion .

On this page, enable and disable postfix templates and appoint the key to activate the template expansion.

Controls

Item	Description
Enable postfix completion	– Select this check box to have IntelliJ IDEA transform expressions with postfixes into other expressions by expanding postfix-specific templates. When the check box is selected, choose the postfixes to apply transformations to by selecting the check boxes next to the desired postfixes in the list below. – When this check box is cleared, no template expansion is applied.
Expand template with	From this drop-down box, choose the key that will invoke template expansion. The available options are: Tab, Space, and Enter.
Table of available postfix templates	The table below shows the list of available postfix templates. To enable or disable a template, select or clear the check box next to it. When you select a template, the right-hand pane shows its description and illustrates how it works by displaying the expression before and after the selected template is applied.

Console

File | Settings | Editor | Editor | Console Folding

Use this page to define lines to be folded in consoles. This lets you hide extraneous information and make console output easier to read and comprehend.

Item	Description
Use soft wraps in console	If this check box is selected, soft wraps (or word wraps) are used in consoles.
Console commands history size	In this text box, specify how many console commands will be included in the console history and can be browsed through.
Override console cycle buffer size (1024 KB)	Select this check box if you want to delete old messages when the console buffer size exceeds the specified value.
Fold console lines that contain	In this area, configure a list of patterns that determine the lines to be hidden in console output. Any line that contains one of these patterns as a substring is hidden. Use the following icons or shortcuts:

Icon	Keyboard shortcut	Description
+	Alt+Insert	Use this icon or shortcut to open the Folding Pattern dialog box and type a new pattern.
	Enter	Use this icon or shortcut to open the Folding Pattern dialog box and edit the selected pattern.
−	Alt+Delete	Use this icon or shortcut to remove the selected pattern from the list.

Exceptions	In this area, configure a list of patterns that determine the lines not to be folded in console output. Any line that contains one of these patterns as a substring is displayed. Use the following icons or shortcuts:

Icon	Keyboard shortcut	Description
+	Alt+Insert	Use this icon or shortcut to open the Folding Pattern dialog box and type a new pattern.
	Enter	Use this icon or shortcut to open the Folding Pattern dialog box and edit the selected pattern.
−	Alt+Delete	Use this icon or shortcut to remove the selected pattern from the list.

Colors and Fonts

File | Settings | Editor | Colors and Fonts for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Colors and Fonts for OS X

Ctrl+Alt+S

On this page:

– Scheme
– Scheme settings

Scheme

Use this section to select the Colors and Fonts scheme. IntelliJ IDEA suggests several pre-defined schemes, one of them being the default.

Warning! The selected scheme applies to the editor only!
– The pre-defined schemes cannot be edited. To change colors and fonts settings in such a scheme, create its copy. Refer to the section Configuring Colors and Fonts for details.

Item	Description
Scheme name	From this drop-down list, select the Colors & Fonts scheme to use in your workspace. When a non-default scheme is selected, the node name changes its color to blue. Warning! The pre-defined schemes cannot be edited. To change colors and fonts settings in such a scheme, create its copy.
Save As	Click this button to save the currently selected Colors & Fonts settings as a new scheme. After saving, the scheme settings become editable.
Delete	Click this button to delete the current scheme.

Scheme settings

Use the corresponding pages to change font type, colors and highlighting for the supported languages , consoles, debugger, version control, differences viewer, file statuses, and scopes. Note that besides color settings, highlighting is determined by your Inspection Profile . For example, if you do not want unused symbols to be highlighted, turn off the Unused Symbol inspection in the profile being used.

Choose specific nested page, and configure as required.

Page	Description
Font	Use this page to define the family of the font to be used in the editor, its size, and line spacing. In the Editor Font area you can configure primary and secondary fonts, their size and line spacing. Warning! The options in the Editor Font section can be modified in the editable schemes only. For the default schemes, all the controls are disabled! Change the following options: – Show only monospaced fonts - By default, this check box is selected and the Primary font list will show only monospaced fonts. To choose among all the fonts available in the system, clear this check box. If an entered character cannot be displayed in the current font, the closest match is used. – Primary font list shows the selection of all the fonts, detected on the computer. – Secondary font list shows the selection of fonts, which will be used if IntelliJ IDEA fails to use a primary one. – Enable font ligatures: select this check box to show the typographic ligatures . Note that this feature depends on the selected fonts.
General	Use this page to customize the font type and colors for the editor textual components, specified in the list.
Console Colors	Use this page to define the colors to be used in the run or debug console (see Run Tool Window , Debug Tool Window. Console).
Console Font	Use this page to define the family of the font to be used in the run or debug console, its size, and line spacing (see Run Tool Window , Debug Tool Window. Console). In the Console Font area you can configure primary and secondary fonts, their size and line spacing. Warning! The options in the Console Font section can be modified in the editable schemes only. For the default schemes, all the controls are disabled! Change the following options: – Show only monospaced fonts - By default, this check box is selected and the Primary font list will show only monospaced fonts. To choose among all the fonts available in the system, clear this check box. If an entered character cannot be displayed in the current font, the closest match is used. – Primary font list shows the selection of all the fonts, detected on the computer. – Secondary font list shows the selection of fonts, which will be used if IntelliJ IDEA fails to use a primary one. – Enable font ligatures: select this check box to show the typographic ligatures . Note that this feature depends on the selected fonts. – Apply editor font settings: click this button to apply editor font settings to the console fonts.
Custom	Use this page to customize the font type and colors in user-defined file types , in particular, for custom keywords highlighting .
VCS	Use this page to customize the colors of the left gutter markers and VCS annotations . To change color of any VCS component, click the corresponding color swatch to show the Color Picker dialog, and select the desired color.
File Status	Use this page to customize font type and colors to visually denote the current file status with regards to a VCS repository .
By Scope	Use this page to customize colors and highlighting for class members assigned to various scopes. To edit a scope, click the Edit Scopes button below the pane with a list of existing scopes. The Scopes dialog box opens.
Languages and IntelliJ	Use these pages to customize font type and colors for the corresponding file types and the IntelliJ IDEA components.

IDEA components

Code Style

File | Settings | Editor | Code Style for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style for OS X

Ctrl+Alt+S

Use this page to choose the code style scheme and change it as required. Settings of a code style scheme are automatically applied every time IntelliJ IDEA generates, refactors, or reformats your code.

On this page:

– Scheme
– Line Separators
– Indents Detection
– Formatter Control
– EditorConfig

Scheme

In this area, choose the code style scheme and change it as required. Settings of a code style scheme are automatically applied every time IntelliJ IDEA generates, refactors, or reformats your code.

The code styles are defined at the project level and at the global level.

– At the Project level, settings are grouped under the Project scheme, which is predefined and is marked in bold font. The Project style scheme applies to the current project only.
  The default scheme cannot be changed! When any modifications are made, IntelliJ IDEA automatically creates a copy of the default scheme. If you want to use a different project level scheme, choose the desired scheme and copy its settings to project.
– At the Global level, settings are grouped under the predefined Default scheme, and any other scheme created by the user. The global settings are used when the user doesn't want to keep code style settings with the project and share them.

Item	Description
Scheme	From this drop-down list, select the scheme to be used. The predefined schemes are shown in bold. The custom schemes, ones created as copies of the predefined schemes, are in plain text.
Manage	Click this button to open the Code Style Schemes dialog that lets you manage the schemes.
Reset	This link appears in the upper-right corner of the dialog when the current style or scheme are modified. Click this button to discard changes.

Line Separators

IntelliJ IDEA lets you configure line separator and indentation options for various languages. When reformatting source code , IntelliJ IDEA will apply the specified indentation behavior and skip the sections denoted with the special formatting off/on markers.

Item	Description
Line Separator (for new files)	Use this drop-down list to specify which line separator is to be used in files created by IntelliJ IDEA. The available options are: – System dependent - choose this option to use the default selection. – Unix and OS X (\n) - choose this option to use the Unix and OS X line separator. – Windows (\r\n) - choose this option to use the Windows line separator. – Classic Mac (\r) - choose this option to use the Classic Mac line separator. Refer to the section Configuring Line Separators .
Right Margin (columns)	In this text box, specify the number of columns to be used to display pages in the editor.
Wrap when typing reaches right margin	Select this check box to ensure that edited text always fits in the specified right margin.

Indents Detection

Use this area to specify default options for indentation.

Item	Description
Detect and use existing file indents for editing	Select this check box for IntelliJ IDEA to detect the existing indents in the file and use them for editing instead of the indents specified in the Code Style settings for the specific language.
Show notifications about detected indents	Select this check box to show a notification if IntelliJ IDEA detects indents that are different from the ones specified in the Code Style settings for the specific language.

Formatter Control

In this area, specify the markers to limit code fragments that you want to exclude from reformatting, see more in Reformatting Source Code . In the source code, formatting markers are written inside line comments, see Commenting and Uncommenting Blocks of Code .

Item	Description
Enable formatter markers in comments	– If this check box is selected, the fragments of code between the line comments with the formatting markers will not be reformatted but will preserve the original formatting. After you select the check box, the fields below become available and you can specify the character strings to be treated as formatting markers. – If the check box is cleared, the formatting markers will be ignored and the code between the line comments with markers will be reformatted.
Markers	
Formatter off:	In this text box, specify the character string that will indicate the beginning of a code fragment which you want to exclude from reformatting. Type a character string

	with the `@` symbol in preposition or leave the predefined value `@formatter:off`.
Formatter on:	In this text box, specify the character string that will indicate the end of a code fragment which you want to exclude from reformatting. Type a character string with the `@` symbol in preposition or leave the predefined value `@formatter:on`.
Regular expressions	Select this check box to use regular expressions instead of specifying the formatting markers explicitly. IntelliJ IDEA matches formatter on/off markers using the regular expression specified instead of the exact string.

Example of using formatting markers

The original source code

```
//@formatter:off
    "scripts": {
        "post-install-cmd": [
            "php artisan optimize"
        ],
        "post-update-cmd": [
            "php artisan clear-compiled",
            "php artisan optimize"
        ],
        "post-create-project-cmd": [
            "php artisan key:generate"
        ]
    },
//@formatter:on
```

The code after reformatting

When the formatting markers are disabled, the original formatting is broken:

```
//@formatter:off
  "scripts": {
      "post-install-cmd": [
          "php artisan optimize"
      ],
      "post-update-cmd": [
          "php artisan clear-compiled",
          "php artisan optimize"
      ],
      "post-create-project-cmd": [
          "php artisan key:generate"
      ]
  },
//@formatter:on
```

When the formatting markers are enabled, the original formatting is preserved:

```
//@formatter:off
    "scripts": {
        "post-install-cmd": [
            "php artisan optimize"
        ],
        "post-update-cmd": [
            "php artisan clear-compiled",
            "php artisan optimize"
        ],
        "post-create-project-cmd": [
            "php artisan key:generate"
        ]
    },
//@formatter:on
```

EditorConfig

In this area enable the support of the EditorConfig plugin.

Item	Description
Enable EditorConfig support	Select this check box to enable the EditorConfig plugin support. In this case you can specify your own code style settings that override the IDE settings. However, if you decide to use IDE settings after creating the EditorConfig settings file then you need clear the Enable EditConfig support check box. See also Configuring Code Style procedure.

Code Style Schemes

File | Settings | Code Style - Manage for Windows and Linux

IntelliJ IDEA | Preferences | Code Style - Manage for OS X

Ctrl+Alt+S

Use this dialog box to manage the set of code style schemes.

Item	Description
Save As...	Click this button to create a copy of the currently selected scheme. This new scheme can be used for copying a scheme to project, and for export .
Delete	Click this button to remove the currently selected scheme. Note that the predefined schemes cannot be deleted.
Copy to Project	Click this button to copy the settings of the currently selected scheme to project. When the settings are copied, IntelliJ IDEA suggests to switch to this scheme.
Import	Click this button to import an IntelliJ IDEA XML code style settings, or JSCS config file.
Export	Click this button to export to an IntelliJ IDEA XML code style settings file.
Close	Click this button to close the Code Style Schemes dialog box.

Note that any changes to the set of schemes (creating copies, or deleting unnecessary schemes) only take place on clicking Apply or OK buttons in the Settings/Preferences dialog .

Code Style. HOCON

File | Settings | Editor | Code Style | HOCON for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | HOCON for OS X

Use this page to configure formatting options for HOCON files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Wrapping and braces
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines

Note This page appears when the Scala plugin is enabled.

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code

when multiline	construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch`.

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line. – Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line. – Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level. – Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces	From this drop-down list, choose the braces introduction method for `if`, `for`, `while`, and `do () while` statements. The available options are:
	– Do not force - select this option to suppress introducing braces automatically.
	– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is
	`<?php`
	`if (true)`
	`return false;`
	IntelliJ IDEA will insert braces automatically:
	`<?php`
	`if (true)`
	`{`
	`return false;`
	`}`
	– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.

Code Style. Scala

File | Settings | Editor | Code Style | Scala for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | Scala for OS X

Use this page to configure formatting options for Scala files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Wrapping and braces
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines
- Imports
- Multi-line strings
- Type Annotations
- Arrangement
- Other

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch`.

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line. – Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line. – Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level. – Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces	From this drop-down list, choose the braces introduction method for `if`, `for`, `while`, and `do () while` statements. The available options are:
	– Do not force - select this option to suppress introducing braces automatically.
	– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is
	`<?php`
	`if (true)`
	`return false;`
	IntelliJ IDEA will insert braces automatically:
	`<?php`
	`if (true)`
	`{`
	`return false;`
	`}`
	– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
Minimum Blank Lines	In the text boxes in this area, specify the number of blank lines to be present in the specified locations. **Warning!** These settings do not influence the number of blank lines before the first and after the last item.

Imports

Use this tab to configure import settings for your project.

Item	Description
Class count to use import with '_':	Use this text field, specify the number of classes to be imported from a single package until all statements importing a single class are substituted with a statement importing an entire package.
Add import statement in closest block	Select this check box to add your import statement to the closest block instead of the top of your code.
Add fully qualified imports	Select this check box to add fully qualified imports.
Do not change path during optimize imports for local imports	Select this check box to eliminate changing the local imports into the fully qualified imports.
Sort imports lexicographically (for optimize imports)	Use this check box if you want IntelliJ IDEA to sort your imports in lexicographical order. **Note** Note that if Add fully qualified imports check box is selected then fully qualified imports are sorted first.
Collect imports with the same prefix into one import	Use this check box if you want IntelliJ IDEA to insert all imports with the same prefix into one.

Import members using '_'	If you select this check box then IntelliJ IDEA imports all members included in the object. If the check box is cleared, IntelliJ IDEA imports only a specific member such as method or a field from the object.
Use the shortest path, when trying to import reference with already imported name	Select this check box to let IntelliJ IDEA find the shortest available path when you import the reference with already imported name using code completion.
Classes to use only with prefix	This area lets you view the list of classes with prefixes. You can add a pattern to use appropriate classes only with prefix or you can remove a class from the list.
Import Layout	this area lets you view the layout of the import statements, add or remove blank lines between them,and add or remove package names. You can also move the rows in the layout up or down.

Multi-line strings

Item	Description

Multi-line support	Use this list to choose the support for the multi-line strings. You can choose from the following options:
	– Insert margin char - this is a default option that lets you add `stripMargin` method.
	`val age = """ | """.stripMargin`
	– Closing quotes on new line - use this option to get quotes on a new line when you press Enter in multi-line strings.
	`val age = """ """`
	– None - use this option if you don't want to use any special support for multi-line strings.
	`val age = """ """`

Margin char	Use this field to enter a character to use in the `stripMargin` method or use the default one (|).
Margin char indent	Use this field to increase or decrease the indent for the Margin char.
Opening quotes on new line	Select this check box to place the opening quotes on the new line.
Keep opening quotes	Select this check box to keep the opening quotes on the same line overriding the Opening quotes on new line option.
Strip/Insert margin on copy/paste	Select this check box to remove the margin character when you copy/paste content of the margin. When this check box is cleared, the margin character is also copied and pasted.

Type Annotations

Use these settings for the code inspections.

Item	Description
Instance Type Annotations	Use this area to configure settings for Instance Type Annotations.
Local type Annotations	Use this area to configure settings for Local Type Annotations.
Special Cases	Use this area to configure setting for special cases in type annotations.

Arrangement

This tab lets you define a set of rules that rearranges your code according to your preferences.

Item	Description
Grouping Rules	Use this area to set the grouping rules. – Keep dependent methods together Select this check box to group the dependent methods together. In order: list, select depth-first or breadth-first options. – Keep_java_style_getters_and_setters_together Select this check box to keep Java getter and setter methods together. By default, this check box is selected. – Keep_scala_style_getters_and_setters_together Select this check box to keep Scala getter and setter methods together. By default, this check box is selected. – Split into unarrangeable blocks by expressions Select this check box
Matching rules	Use this area to define elements order as a list of rules, where every rule has a set of matches such as modifier or type. – - use this button to add a rule. The empty rule dialog window opens. – - use this button to remove the rule from the list. – - use this button to edit an existing rule. To see this button, navigate to the rule that you want to edit and click on the button. In pop-up window that opens, modify the rule fields. – - use these buttons to move the selected rule up or down.
Empty rule	Use this window to create a new matching rule or edit an existing one. You can select from the following filters: – Type - use this filter to choose classes or methods for your rule. Note that clicking twice on the type keyword will negate the condition. – Modifier - use this filter to select the types of modifiers for the rule. Note that clicking twice on the modifier keyword will negate the condition. – Order - use this option to select the sorting order for the rule. This option is useful when more than one element uses the same matching rule. In this case, selecting Keep order will keep the same order as was set before the rearrangement and selecting Order by Name will sort the elements with the same matching rule by their names.
	This icon appears when you select Order by Name in Order option. The icon indicates that the items in this rule are alphabetized.

Other

Use this tab to configure syntax when you use certain symbols in the code.

Item	Description
Enforce procedure syntax for methods with Unit return type	Select this check box to show the explicit syntax for methods with Unit return type.
Replace '=>' with unicode symbol	Select this check box to change '=>' symbol to one in Unicode.
Replace '->' with unicode symbol	Select this check box to change '->' symbol to one in Unicode.
Replace '<-' in "for" generator with unicode symbol	Select this check box to change '<-' symbol to one in Unicode.
Kind Projector: Replace 'Lambda' with unicode symbol	Select this check box to change 'Lambda' symbol to one in Unicode.

Code Style. Java

File | Settings | Editor | Code Style | Java for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | Java for OS X

Use this page to configure formatting options for Java files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Wrapping and braces
  - Right Margin (columns)
  - Wrap on typing
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines
- JavaDoc
- Arrangement
- Imports
- Code Generation
- Java EE Names

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.
Label indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted at the next line before a label statement.
Absolute label indent	If this check box is selected, label indentation is counted as an absolute number of spaces. Otherwise, label indentation is counted relative to previous indent levels.
Do not indent top level class members	Select this check box to have top level class members located at the class declaration indentation level.
Use indents relative to expression start	Use this check box to switch between the two possible indentation behaviors: – If this check box is not selected, the blocks of code will be formatted against the closest ancestor block that starts on a new line. – If this check box is selected, the blocks of code will be formatted in columns.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

– Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings .

– Yes - in this case IntelliJ IDEA uses the value specified in the Right Margin field.

– No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch` .

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line. – Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line. – Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level. – Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces	From this drop-down list, choose the braces introduction method for `if` , `for` , `while` , and `do () while` statements. The available options are:
	– Do not force - select this option to suppress introducing braces automatically.
	– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is
	`<?php`
	`if (true)`
	`return false;`
	IntelliJ IDEA will insert braces automatically:
	`<?php`
	`if (true)`
	`{`
	`return false;`
	`}`
	– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
Minimum Blank Lines	In the text boxes in this area, specify the number of blank lines to be present in the specified locations. Warning! These settings do not influence the number of blank lines before the first and after the last item.

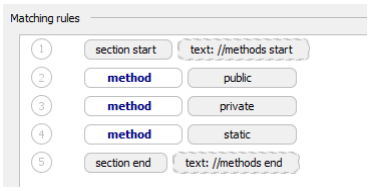
JavaDoc

Item	Description
Alignment	In this area, define how JavaDoc comments should be aligned. – Align parameter description: select this check box to have parameter descriptions aligned against the longest parameter name. Otherwise, the description is separated from the corresponding parameter name by a single space. – Align thrown exception description: select this check box to have thrown exception descriptions aligned against the longest exception name. Otherwise, the description is separated from the exception name by a single space.
Blank Lines	In this area, define where blank lines should be inserted in JavaDoc comments. – After description: select this check box to have a blank line automatically inserted after the description section of a JavaDoc comment. – After parameter descriptions: select this check box to have a blank line inserted after the group of `@param` tags. – After return tag: select this check box to have a blank line inserted after the `@return` tag.
Invalid Tags	In this area, define whether invalid tags should be preserved or not. – Keep invalid tags: select this check box to have the `@invalidTag` preserved. – Keep empty @param tags: select this check box to have `@param` tags without description preserved. – Keep empty @return tags: select this check box to have `@return` tags without description preserved. – Keep empty @throws tags: select this check box to have `@throws` tags without description preserved.
Other	In this area, specify additional formatting options for JavaDoc comments. – Enable leading asterisks: select this check box to have each line of a JavaDoc comment start with an asterisk. – Use @throws rather than @exception: select this check box to have the `@throws` tag used. – Wrap at right margin: select this check box to have the text that exceeds the right margin wrapped to the next line. – Generate `</p>` on empty lines: select this check box to have a `</p>` tag automatically inserted in an empty line. – Keep empty lines: select this check box to have manually added empty lines preserved. – Do not wrap one-line comment: select this check box to have short comments kept in one line with the opening and closing tags. – Preserve line feed: If this check box is not selected (by default), line feeds are not preserved on reformatting. This is convenient when comments should be formatted within the boundaries of a paragraph, to occupy minimum space. If this check box is selected, line feeds will be preserved. – Parameter description on new line: select this check box to instruct the IntelliJ IDEA formatter to place the description of a JavaDoc parameter (if any) to a new line. It uses indent based on the continuation indent value.

Arrangement

This tab lets you define a set of rules that rearranges your code according to your preferences.

Item	Description
Grouping Rules	Use this area to set the grouping rules. – Keep getters and setters together Select this check box to keep getter and setter methods together. By default, this check box is selected. – Keep overridden methods together Select this check box to group the overridden methods together by class and interface. In order: list, select keep or by name options. – Keep dependent methods together Select this check box to group the dependent methods together. In order: list, select depth-first or breadth-first options.

Matching rules	Use this area to define elements order as a list of rules, where every rule has a set of matches such as modifier or type.
	– + - use this button to add a rule. The empty rule dialog window opens.
	– +S - use this button to add a section rule. The section rule lets you move methods or variables into sections that you have defined.
	
	For example, you can create the following section rule:
	
	 
	
	After the arrangement, methods in the class will be rearranged as specified in the created section rule and will be surrounded by comments:
	
	`//methods start`
	`public void test() {}`
	`private int a() {`
	`return 1; }`
	`static void r() {}`
	`//methods end`
	
	– - use this button to remove the rule from the list.
	– - use this button to edit an existing rule. To see this button, navigate to the rule that you want to edit and click on the button. In pop-up window that opens, modify the rule fields.
	– - use these buttons to move the selected rule up or down.
	– - use this button to configure an alias for the matching rule. In this case, when you create an arrangement rule you can define a custom rule (alias) that would include a sequence of different rules and apply the alias to your current rule.

Empty rule	Use this window to create a new matching rule or edit an existing one. You can select from the following filters:

- Type - use this filter to choose classes or methods for your rule.

  Note that clicking twice on the type keyword will negate the condition.
- Modifier - use this filter to select the types of modifiers for the rule.

  Note that clicking twice on the modifier keyword will negate the condition.
- Name - use this field to specify entry names in the rule. This filter matches only entry names such as field names, method names, class names and etc. The filter supports regular expressions and uses a standard syntax. The match is performed against the entire name.
- Order - use this drop-down list to select the sorting order for the rule. This option is useful when more than one element uses the same matching rule. In this case, selecting Keep order will keep the same order as was set before the rearrangement and selecting Order by Name will sort the elements with the same matching rule by their names.
- Aliases - this option displays aliases that were defined in the Rules Alias Definition dialog. You can remove the ones you do not need.

This icon appears when you select Order by Name in Order option. The icon indicates that the items in this rule are alphabetized.

Imports

This table lists actions to be performed when imports are optimized .

Item	Description
General	In this area, configure general import options. Options: – Use single class import: Select this check box to have IntelliJ IDEA import only a particular class from a package during code generation or import optimization . Otherwise, a statement importing an entire package is inserted. – Use fully qualified class names: Select this check box to have IntelliJ IDEA use the fully qualified name of the class to be imported during code generation or import optimization . Otherwise, a normal import statement is inserted. – Insert imports for inner classes: Select this check box to have IntelliJ IDEA create imports for the inner classes referenced in your code. – Use fully qualified names in Javadoc: Select this check box to have IntelliJ IDEA use a fully qualified class name in Javadoc. Otherwise, a class is imported. – Class count to use import with '*': In this text field, specify the number of classes to be imported from a single package until all statements importing a single class are substituted with a statement importing an entire package. – Names count to use static import with '*': In this text box, specify the number of members to be imported from a single class until all statements importing a single member are substituted with a statement importing an entire class.
JSP Imports Layout	In this area, configure how JSP import statements should be organized in your code. The introduced changes are displayed in the Preview pane below. Options: – Prefer comma separated import list: Select this option to import statements organized in a comma separated list. – Prefer one import statement per page directive: Select this option to have one import statement created per line.
Packages to Use Import with '*'	In this area, configure a list of packages and classes to be always imported completely. Options: – Static: Select this check box, if you want to declare static import for the selected class. – Package: In the text fields of this column, specify the packages and classes to be always imported completely. – With Subpackages: Select this check box to have all the subpackages of the selected package imported completely. – Add Package: Click this button to add a new entry to the list of packages and classes. – Add Blank: Click this button to add an empty separator to the list of packages and classes. – Remove: Click this button to delete the selected package or class from the list.
Import Layout	In this area, configure how import statements should be organized in your code. You can set up certain classes to be positioned first, or last, or one after another. Imported classes will be grouped as per their packages and sorted alphabetically within a package. Options: – Layout static imports separately: If this check box is selected, all static imports will be kept in a separate section. Otherwise, all import statements will be sorted according to the specified layout rules. – Static: Select this check box, if you want to declare static import for the selected package. – Package: In the text fields of this column, specify the packages to be imported. – With Subpackages: Select this check box to have IntelliJ IDEA apply the layout rules to all the subpackages of the selected package. – Add Package: Click this button to add a new entry to the list of packages. – Add Blank: Click this button to have a blank line inserted after the selected entry, which indicates that a blank line should be inserted between the corresponding import statements. – Move Up / Move Down: Click these buttons to move a package or a blank line up or down in the list thus defining the order of import statements. – Remove: Click this button to delete the selected package from the list.

Code Generation

Item	Description
Naming	Options: – Prefer longer names: select this check box to have the longest name highlighted in a lookup list for code completion. Otherwise, the shortest name is highlighted. – Name prefix / suffix: in these text boxes, type the prefixes and suffixes to be used when generating suggestions for naming new symbols through the IntelliJ IDEA code-generation features. If the fields are left blank, then the default name suggestions without prefixes or suffixes will be used. When you add a prefix value, IntelliJ IDEA automatically converts the first letter of the suggested base name to upper case. For example, if the prefix for a static field is defined as `s`, and the type of the field is `Counter`, then the suggested static field name will be `sCounter`. Note that this prefix will not take part in the generation of the getter and setter method names. So, in our example, the accessor names will be `getCounter` and `setCounter` respectively. Specify the name prefixes and suffixes for fields, static fields, parameters, and local variables.
Final Modifier	Options: – Make generated local variables final - select this check box to have local variables in the IntelliJ IDEA-generated code supplied with final modifiers. – Make generated parameters final - select this check box to have parameters in the IntelliJ IDEA-generated code supplied with final modifiers.

Comment Code	Options: – Line comment at first column - select this check box to have generated line comments placed in the first column. – Block comment at first column - select this check box to have generated block comments placed in the first column.
Order of Members	Specify the necessary order: – Order of Members - the list defines the order in which code elements appear when IntelliJ IDEA inserts them by itself (for instance, in case of Intention Actions). – Move Up/Move Down - Use these buttons to change the order of members on the list and thus re-define the order in which corresponding elements appear in the generated code.
Other Controls	Select the necessary options: – Use External Annotations - if this check box is selected, IntelliJ IDEA will prompt to specify whether you want an annotation to be stored in the source code or externally . Otherwise, if the check box is cleared, annotations are added to the source code by default. – Insert @Override Annotation - Select this check box to have IntelliJ IDEA insert @Override annotations automatically.

Java EE Names

Use this tab to specify prefixes and suffixes for the names of bean classes, bean interfaces, `<ejb-name>` tags, servlets, filters, and listeners. These prefixes and suffixes will by default appear in the corresponding fields of the New Bean , New Servlet , New Filter , and New Listener dialog boxes. For entity beans, you can also specify the default primary key class that appears in the Create CMP Field dialog box.

Item	Description
Entity Bean	Use the text boxes in this area to specify prefixes and/or suffixes to be used in the names of Entity Bean components: – EJB Class: default suffix `Bean` . – Home Interface: default suffix `Home` . – Remote Interface – Local Home Interface: default prefix `Local` , default suffix `Home` . – Local Interface: default prefix `Local` . – <ejb-name> tag: default suffix `EJB` . – Transfer Object: default suffix `VO` . – Default PK Class: `java.lang.String` .
Session Bean	Use the text boxes in this area to specify prefixes and/or suffixes to be used in the names of Session Bean components: – EJB Class: default suffix `Bean` . – Home Interface: default suffix `Home` . – Remote Interface – Local Home Interface: default prefix `Local` , default suffix `Home` . – Local Interface: default prefix `Local` . – Service Endpoint Interface: default suffix `Service` . – <ejb-name> tag: default suffix `EJB` .
Message Driven Bean	Use the text boxes in this area to specify prefixes and/or suffixes to be used in the names of Message Driven Bean components: – EJB Class: default suffix `Bean` . – <ejb-name> tag: default suffix `EJB` .
Servlet	Use the text boxes in this area to specify prefixes and/or suffixes to be used in the names of Servlets: – Servlet Class – <servlet-name> tag
Filter	Use the text boxes in this area to specify prefixes and/or suffixes to be used in the names of Filters: – Filter Class – <filter-name> tag
Listener	In this area, specify the prefix and suffix to be used in the names of listener classes.

Code Style. ActionScript

File | Settings | Editor | Code Style | ActionScript for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | ActionScript for OS X

Use this page to configure formatting options for ActionScript files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Wrapping and braces
  - Right Margin (columns)
  - Wrap on typing
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines
- Other
- Arrangement

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings .
- Yes - in this case IntelliJ IDEA uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description

Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch`.

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line. – Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line. – Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level. – Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces	From this drop-down list, choose the braces introduction method for `if`, `for`, `while`, and `do () while` statements. The available options are:
	– Do not force - select this option to suppress introducing braces automatically.
	– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is
	`<?php`
	`if (true)`
	`return false;`
	IntelliJ IDEA will insert braces automatically:
	`<?php`
	`if (true)`
	`{`
	`return false;`
	`}`
	– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
Minimum Blank Lines	In the text boxes in this area, specify the number of blank lines to be present in the specified locations. **Warning!** These settings do not influence the number of blank lines before the first and after the last item.

Other

Item	Description
Indent package statement children	Select this check box to have the nested statements of package statements indented on code reformatting.
Align C-style comments /*...*/	Select this check box to adjust lines in C-style comments.
Align object properties	From the drop-down list, select the type of objects' alignment: – Do not align: the attributes in sequential lines will be not aligned. – On colon: the attributes in sequential lines will be aligned against the colon. – On value: the attributes in sequential lines will be aligned against the value.

Field prefix	In this text box, specify the prefix to be used in ActionScript fields.
Property prefix	In this text box, specify the prefix to be used in ActionScript properties.
Use semicolon to terminate statements	Select this check box to have statements terminated with a semicolon.
Spaces before type reference colon ':'	Select this check box to separate type reference with the space before colon.
Spaces after type reference colon ':'	Select this check box to separate type reference with the space after colon.

Arrangement

This tab lets you define a set of rules that rearranges your code according to your preferences.

Item	Description
Grouping Rules	Use this area to set the grouping rules. – Group property field with corresponding getter/setter
Matching rules	Use this area to define elements order as a list of rules, where every rule has a set of matches such as modifier or type. – - use this button to add a rule. The empty rule dialog window opens. – - use this button to remove the rule from the list. – - use this button to edit an existing rule. To see this button, navigate to the rule that you want to edit and click on the button. In pop-up window that opens, modify the rule fields. – - use these buttons to move the selected rule up or down.
Empty rule	Use this window to create a new matching rule or edit an existing one. You can select from the following filters: – Type - use this filter to choose classes or methods for your rule. Note that clicking twice on the type keyword will negate the condition. – Modifier - use this filter to select the types of modifiers for the rule. Note that clicking twice on the modifier keyword will negate the condition. – Name - use this field to specify entry names in the rule. This filter matches only entry names such as field names, method names, class names and etc. The filter supports regular expressions and uses a standard syntax. The match is performed against the entire name. – Order - use this drop-down list to select the sorting order for the rule. This option is useful when more than one element uses the same matching rule. In this case, selecting Keep order will keep the same order as was set before the rearrangement and selecting Order by Name will sort the elements with the same matching rule by their names. – Aliases - this option displays aliases that were defined in the Rules Alias Definition dialog. You can remove the ones you do not need.
	This icon appears when you select Order by Name in Order option. The icon indicates that the items in this rule are alphabetized.

Code Style. CFML

File | Settings | Editor | Code Style | CFML for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | CFML for OS X

Use this page to configure formatting options for CFML files. View the result in the Preview pane on the right.

On this page:

- Spaces
- Wrapping and braces
  - Right Margin (columns)
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines
- Set from...

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings is used.

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch` .

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line.

- Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line.
- Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level.
- Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces

From this drop-down list, choose the braces introduction method for `if`, `for`, `while`, and `do () while` statements. The available options are:

- Do not force - select this option to suppress introducing braces automatically.
- When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is

```
<?php
                    if (true)
                    return false;
```

IntelliJ IDEA will insert braces automatically:

```
<?php
                    if (true)
                    {
                    return false;
                    }
```

- Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Set from...

Click this link to choose the base for the current language default code style from the pop-up list, that appears. The list contains two options:

- Language: choose this option to inherit the coding style settings from another language. Select the source language from the list, that opens. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.
- Predefined code style: choose this option to use the coding standards defined for a specific framework. Select one of the following frameworks from the list:
  - PEAR
  - Zend
  - PSR1 / PSR2
  - Symfony2
  - Joomla!

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. CoffeeScript

File | Settings | Editor | Code Style | CoffeeScript for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | CoffeeScript for OS X

Use this page to configure formatting options for CoffeeScript files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Wrapping and braces
  - Right Margin (columns)
  - Wrap on typing
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines
- Other
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings .
- Yes - in this case IntelliJ IDEA uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch`.

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line. – Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line. – Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level. – Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces	From this drop-down list, choose the braces introduction method for `if`, `for`, `while`, and `do () while` statements. The available options are:
	– Do not force - select this option to suppress introducing braces automatically.
	– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is
	`<?php`
	`if (true)`
	`return false;`
	IntelliJ IDEA will insert braces automatically:
	`<?php`
	`if (true)`
	`{`
	`return false;`
	`}`
	– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Other

Item	Description
Align object properties	From the drop-down list, select the type of objects' alignment: – Do not align: the attributes in sequential lines will be not aligned. – On colon: the attributes in sequential lines will be aligned against the colon. – On value: the attributes in sequential lines will be aligned against the value.
Line comments at first column	Select this check box to place a line comment in the first column.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. CSS

File | Settings | Editor | Code Style | CSS for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | CSS for OS X

Use this page to configure formatting options for CSS files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Other
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Other

In this tab, specify the alignment, braces and spaces options to be applied on reformatting.

Item	Description
Braces placement	Use this drop-down list to specify where IntelliJ IDEA should place the opening braces of selectors. The available options are: – At the end of line – Next line
Align values	Use this drop-down list to specify how IntelliJ IDEA should align attributes and values. The available options are: – Do not align: select this option to specify alignment on the first character of an attribute name. – On value: select this option to specify alignment on the first character of the value of an attribute. – On colon
Blank lines between blocks	In this text box, specify the minimum number of sequential blank lines to be retained after reformatting.
Align closing brace with properties	If this check box is selected, the closing brace of the selector will be placed under the list of properties. If this check box is not selected, the closing brace of the selector will be placed under the selector.
Keep single-line blocks	If this check box is selected, the blocks with a single property will be confined to one line. If this check box is not selected, each property will be placed to its own line.
Spaces	Select the check boxes in this area to add a space after the colon delimiting key and value, and before the opening brace of the selector.
HEX Colors	Use this area to configure the hex color syntax. You can select from the following check options: – Convert hex colors to - select this check box to configure the hex color letter case. You can choose Lower case or Upper case. – Convert hex colors format to - select this check box to configure the hex color format length. You can choose Long format or Short format. View changes in the Preview pane.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. Gherkin

File | Settings | Editor | Code Style | Gherkin for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | Gherkin for OS X

Use this page to configure formatting options for Gherkin files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. Groovy

File | Settings | Editor | Code Style | Groovy for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | Groovy for OS X

Use this page to configure formatting options for Groovy files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Wrapping and braces
  - Right Margin (columns)
  - Wrap on typing
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines
- Imports
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.
Label indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted at the next line before a label statement.
Absolute label indent	If this check box is selected, label indentation is counted as an absolute number of spaces. Otherwise, label indentation is counted relative to previous indent levels.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings .
- Yes - in this case IntelliJ IDEA uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch`.

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line. – Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line. – Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level. – Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces	From this drop-down list, choose the braces introduction method for `if`, `for`, `while`, and `do () while` statements. The available options are:
	– Do not force - select this option to suppress introducing braces automatically.
	– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is
	`<?php`
	`if (true)`
	`return false;`
	IntelliJ IDEA will insert braces automatically:
	`<?php`
	`if (true)`
	`{`
	`return false;`
	`}`
	– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
Minimum Blank Lines	In the text boxes in this area, specify the number of blank lines to be present in the specified locations. Warning! These settings do not influence the number of blank lines before the first and after the last item.

Imports

This table lists actions to be performed when imports are optimized .

Item	Description
General	In this area, configure general import options. Options:

- Use single class import: Select this check box to have IntelliJ IDEA import only a particular class from a package during code generation or import optimization . Otherwise, a statement importing an entire package is inserted.
- Use fully qualified class names: Select this check box to have IntelliJ IDEA use the fully qualified name of the class to be imported during code generation or import optimization . Otherwise, a normal import statement is inserted.
- Insert imports for inner classes: Select this check box to have IntelliJ IDEA create imports for the inner classes referenced in your code.
- Use fully qualified names in Javadoc: Select this check box to have IntelliJ IDEA use a fully qualified class name in Javadoc. Otherwise, a class is imported.
- Class count to use import with '*': In this text field, specify the number of classes to be imported from a single package until all statements importing a single class are substituted with a statement importing an entire package.
- Names count to use static import with '*': In this text box, specify the number of members to be imported from a single class until all statements importing a single member are substituted with a statement importing an entire class.

JSP Imports Layout	In this area, configure how JSP import statements should be organized in your code. The introduced changes are displayed in the Preview pane below. Options: – Prefer comma separated import list: Select this option to import statements organized in a comma separated list. – Prefer one import statement per page directive: Select this option to have one import statement created per line.
Packages to Use Import with '*'	In this area, configure a list of packages and classes to be always imported completely. Options: – Static: Select this check box, if you want to declare static import for the selected class. – Package: In the text fields of this column, specify the packages and classes to be always imported completely. – With Subpackages: Select this check box to have all the subpackages of the selected package imported completely. – Add Package: Click this button to add a new entry to the list of packages and classes. – Add Blank: Click this button to add an empty separator to the list of packages and classes. – Remove: Click this button to delete the selected package or class from the list.
Import Layout	In this area, configure how import statements should be organized in your code. You can set up certain classes to be positioned first, or last, or one after another. Imported classes will be grouped as per their packages and sorted alphabetically within a package. Options: – Layout static imports separately: If this check box is selected, all static imports will be kept in a separate section. Otherwise, all import statements will be sorted according to the specified layout rules. – Static: Select this check box, if you want to declare static import for the selected package. – Package: In the text fields of this column, specify the packages to be imported. – With Subpackages: Select this check box to have IntelliJ IDEA apply the layout rules to all the subpackages of the selected package. – Add Package: Click this button to add a new entry to the list of packages. – Add Blank: Click this button to have a blank line inserted after the selected entry, which indicates that a blank line should be inserted between the corresponding import statements. – Move Up / Move Down: Click these buttons to move a package or a blank line up or down in the list thus defining the order of import statements. – Remove: Click this button to delete the selected package from the list.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. GSP

File | Settings | Editor | Code Style | GSP for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | GSP for OS X

Use this page to configure formatting options for GSP files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Set from...

Tabs and Indents

Item	Description

Use tab character	- If this check box is selected, tab characters are used:
	  - On pressing the `Tab` key
	  - For indentation
	  - For code reformatting
	- When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.

Smart tabs	- If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. - If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. HAML

File | Settings | Editor | Code Style | HAML for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | HAML for OS X

Use this page to configure formatting options for HAML files. View the result in the Preview pane on the right.

On this page:

– Tabs and Indents

– Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. HTML

File | Settings | Editor | Code Style | HTML for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | HTML for OS X

Use this page to configure formatting options for HTML files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Other
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Other

Item	Description
Right Margin	Use these settings to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings will be used.
Wrap on typing	Use these settings settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options: – Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings . – Yes - in this case the value in the specified right margin is used. – No - in this case this option is switched off and a line can exceed the number that is specified in the right margin.
Keep line breaks	Select this check box to have IntelliJ IDEA honor line breaks when reviewing HTML files in the editor.
Keep line breaks in text	Select this check box to have IntelliJ IDEA honor line breaks in attributes (for example, lengthy descriptions) when reviewing HTML files in the editor.
Keep blank lines	In this text box, specify the minimum number of sequential blank lines to be retained after reformatting.
Wrap attributes	Use this drop-down list to determine how attribute lines should be wrapped. The available options are: – Do not wrap - if this option is selected, no special wrapping style is applied to the code. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped to give one element per line with proper indentation. – Wrap always - select this option to have all elements in lists wrapped to give one element per line with proper indentation.
Wrap text	Select this check box to have long lines wrapped according to the code style settings.
Align attributes	Select this check box to have attributes in sequential lines aligned.
Align text	Select this check box to have IntelliJ IDEA align the text that occupies several lines within a tag.
Keep white spaces	Select this check box to suppress replacing actual white spaces with tabs.
Spaces	In this area, define the use of spaces for attributes and tag names. – Around "=" in attribute - select this check box to have spaces added around the "=" symbol in attributes. – After tag name - select this check box to have spaces added after tag names. – In empty tag - select this check box to have spaces added in empty tags.
Insert new line before	This display field shows a list of tags before which a new line should be inserted. Use the button next to the field or press `Shift+Enter` to open the Insert New Line Before Tags dialog box, where you can edit the list of tags.

Remove new line before	This display field shows a list of tags before which a break line should be removed. Use the button next to the field or press Shift+Enter to open the Remove Line Breaks Before Tags dialog box, where you can edit the list of tags.
Do not indent children of	This display field shows a list of tags whose children should not be indented. Use the button next to the field or press Shift+Enter to open the Do Not Indent Children Of dialog box, where you can edit the list of tags.
Or if tag size more than	In this text box, specify the minimum length of a tag in lines starting from which its children are not indented.
Inline elements	This display field shows a list of tags that are presented in the source code in the same line with the other tags. If a tag is removed from the list, the editor automatically moves it to a new line, when you add such tag to the source code. Use the button next to the field or press Shift+Enter to open the Inline Elements dialog box, where you can edit the list of tags.
Keep white spaces inside	This display field shows a list of tags inside which you want the editor to preserve white spaces *as is*, without any changes. Use the button next to the field or press Shift+Enter to open the Keep Whitespaces Inside dialog box, where you can edit the list of tags.
Don't break if inline content	This display field shows a list of tags that are not to be wrapped if their content is inlined. Use the button next to the field or press Shift+Enter to open the Don't Wrap If Inline Content Only dialog box, where you can edit the list of tags.
Generated quote marks	Choose the style of the quote marks (double, single, or none) to be automatically inserted around HTML attributes on typing =. This is important when HTML is inserted dynamically using JavaScript or PHP and you want to consistently use double-quote pairs for JavaScript or PHP strings and single-quote pairs for HTML to prevent problems, for example, when copying and pasting.
Enforce on format	If this check box is selected, then on code reformatting the previously generated quote marks will be replaced (for example, double quotes with single quotes).

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. JavaScript

File | Settings | Editor | Code Style | JavaScript for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | JavaScript for OS X

Use this page to configure formatting options for JavaScript files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Wrapping and braces
  - Right Margin (columns)
  - Wrap on typing
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines
- Other
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Indent chained methods	Use this check box to configure indentation of chained method calls.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings .
- Yes - in this case IntelliJ IDEA uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch`.

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line. – Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line. – Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level. – Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces	From this drop-down list, choose the braces introduction method for `if`, `for`, `while`, and `do () while` statements. The available options are:
	– Do not force - select this option to suppress introducing braces automatically.
	– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is
	`<?php`
	`if (true)`
	`return false;`
	IntelliJ IDEA will insert braces automatically:
	`<?php`
	`if (true)`
	`{`
	`return false;`
	`}`
	– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Other

Item	Description
Align object properties	From the drop-down list, select the type of objects' alignment: – Do not align: the attributes in sequential lines will be not aligned. – On colon: the attributes in sequential lines will be aligned against the colon. – On value: the attributes in sequential lines will be aligned against the value.
Use semicolon to terminate statements	Select this check box to have statements terminated with a semicolon.
Generated Code	In this area, configure the code style in `use strict` statements, is you have selected the Prefer Strict Mode check box on the JavaScript page of the

Settings dialog box.

- String literal type: from this drop-down list, choose the type of quotes to be used in `use strict` statements. The available options are Single quotes and Double quotes.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. JSON

File | Settings | Editor | Code Style | JSON for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | JSON for OS X

Use this page to configure formatting options for JSON files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Blank lines
- Wrapping and braces
  - Right Margin (columns)
  - Wrap on typing
  - Keep when reformatting
  - Wrapping options
  - Alignment options
- Other

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.
Indent chained methods	Use this check box to configure indentation of chained method calls.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings .
- Yes - in this case IntelliJ IDEA uses the value specified in the Right Margin field.

– No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected.

If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
Ensure right margin is not exceeded	If this check box is selected, the formatter will do its best to avoid having document lines exceeding the right margin. This option takes precedence over the Do not wrap wrapping style.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch` .

Other

Item	Description
Align object properties	From the drop-down list, select the type of objects' alignment: – Do not align: the attributes in sequential lines will be not aligned. – On colon: the attributes in sequential lines will be aligned against the colon. – On value: the attributes in sequential lines will be aligned against the value.

Code Style. JSP

File | Settings | Editor | Code Style | JSP for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | JSP for OS X

Use this page to configure formatting options for JSP files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Wrapping
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Wrapping

Use this tab to configure wrapping options.

Item	Description
Right Margin	Use these settings to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings will be used.
Wrap on typing	Use these settings settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options: – Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings . – Yes - in this case the value in the specified right margin is used. – No - in this case this option is switched off and a line can exceed the number that is specified in the right margin.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. JSPX

File | Settings | Editor | Code Style | JSPX for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | JSPX for OS X

Use this page to configure formatting options for JSPX files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Wrapping
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Wrapping

Use this tab to configure wrapping options.

Item	Description
Right Margin	Use these settings to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings will be used.
Wrap on typing	Use these settings settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options: – Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings . – Yes - in this case the value in the specified right margin is used. – No - in this case this option is switched off and a line can exceed the number that is specified in the right margin.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. Less

File | Settings | Editor | Code Style | Less for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | Less for OS X

Use this page to configure formatting options for Less files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. PHP
File | Settings | Editor | Code Style | PHP for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | PHP for OS X

Use this page to configure formatting options for PHP files. View the result in the Preview pane on the right.

On this page:

- Set from...
- PHP-secific formatting settings for Tabs and Indents
- PHP-specific formatting settings for Spaces
- PHP-specific formatting settings for Wrapping and Braces
- Blank lines
- PHPDoc
- Other
- Arrangement

Set from...

Click this link to choose the base for the current language default code style from the pop-up list, that appears. The list contains two options:

- Language: choose this option to inherit the coding style settings from another language. Select the source language from the list, that opens. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.
- Predefined code style: choose this option to use the coding standards defined for a specific framework. Select one of the following frameworks from the list:
  - PEAR
  - Zend
  - Symfony2
  - PSR1 / PSR2
  - WordPress
  - Drupal
  - Joomla!

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

PHP-secific formatting settings for Tabs and Indents

See Common Code Style Options: Tabs and Indents for the settings that are common for all languages.

Item	Description
Indent as template language unless PHP-only	– When this check box is selected, IntelliJ IDEA applies the indentation style specified for the template language of the source code into which PHP fragments are embedded. If the PHP code is not surrounded with any other code, the PHP style is applied depending on the status of the Use tab character check box. – When this check box is cleared, the PHP style is applied no matter whether the PHP code fragment is embedded or the entire file is `.php`.

PHP-specific formatting settings for Spaces

See Common Code Style Options: Spaces for the settings that are common for all languages.

Select or clear the check boxes to have spaces inserted, not inserted, or removed in the following PHP contexts:

Item	Description

Before Parentheses	– Array initializer parentheses
	Select this check box to have a space inserted before the opening parenthesis in array declarations, such as:
	`$array = array (0 => "zero", 1 => "one");`
	If the check box is cleared, no space is inserted and the code looks as follows:
	`$array = array(0 => "zero", 1 => "one");`

Around Operators	– Concatenation Select this check box to have spaces inserted before and after the concatenation operator (`.`) so the code looks as follows: `echo "The result is " . $i;` When the check box is cleared, no spaces are inserted around the concatenation operator so the code looks as follows:

```
echo "The result is ".$i;
```

Before Left Brace

– Class left brace
Select this check box to have a space inserted between the class name and the opening brace in class declarations:

```
class Class1 {
                        function Foo()
                    }
```

When the check box is cleared, no space is inserted:

```
class Class1{
                        function Foo()
                    }
```

Selecting or clearing the check box is relevant only when Braces placement in class declarations is set to End of line on the Wrapping and Braces tab.

In Ternary Operator (`?:`)

Select the check boxes in this section to have spaces automatically inserted before and after `?`, before and after `:`, and between `?` and `:` in ternary (conditional) operators.

Other

Select the check boxes in this section to have spaces automatically inserted before and after commas, semicolons, unary NOT operators (`!`), and after type casts.

– After type cast
When this check box is selected, IntelliJ IDEA automatically inserts a space after the closing parentheses of a cast: `$fst = (string) $foo;`

If the check box is cleared, no spaces is inserted and the casted variable sticks to the cast: `$fst = (string)$foo;`

PHP-specific formatting settings for Wrapping and Braces

See Common Code Style Options: Wrapping and Braces for the settings that are common for all languages.

Item **Description**

Braces placement

In this section, choose the position for opening braces in declarations of namespaces, classes, and functions, in loops, and in other constructions. Choose the required position from the drop-down list, the available options are:

– End of line - select this option to have the opening brace placed at the declaration line end.
– Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line.
– Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line.
– Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level.
– Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Extends/implements list

In this section, configure wrapping and subsequent alignment of extend and implements lists:

– Do not wrap - when this option is selected, no special wrapping style is applied.
With this option selected, the nested alignment and braces settings are ignored.
– Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.
– Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.
– Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
– Align when multiline
When this check box is selected, each item in an extends or an implements list, which after wrapping starts on a new line, are aligned by the first item, which remains unwrapped:

```
abstract class Foo extends
                        Class1 implements
                        Class2,
                        Class8 {
                    }
```

When the check box is cleared, the position of each item in a wrapped list is determined by the chosen indentation level.

The status of the check box affects the formatting only if you have chosen to wrap lists, if the Do not wrap option is chosen, selecting or clearing the check box has no effect.

Extends/implements keyword

In this section, configure wrapping for the extends and implements keywords in class declarations.
If you choose the Do not wrap option, no wrapping will be applied, if you choose Wrap if long or Wrap always, each keyword and each item in an extends or implements list will be displayed on a new line:

```
abstract class Foo
                        extends
                        Class1
                        implements
                        Class2,
                        Class8 {
                    }
```

Function declaration parameters

In this area, configure formatting in declarations of functions and methods.

– Keep ')' and '{' on one line
When this check box is cleared, the opening curly brace is moved to the next line:

```
function Foo()
                        {
                        }
```

When the check box is selected, the opening curly brace is displayed on the same line as the function parameters:

```
function Foo() {
                    }
```

Chained method calls

In this section, configure wrapping and subsequent alignment of chained calls.

– Do not wrap - when this option is selected, no special wrapping style is applied.
With this option selected, the nested alignment and braces settings are ignored.
– Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.
– Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.
– Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
– Align when multiline
When the check box is selected, each called method, which after wrapping is positioned on a new line, is aligned by the first one, which remains unwrapped:

```
$x = $x->one( "a", "b" )
                            ->two( "c", "d", "e" )
                            ->three( "fg" )
                            ->four;
```

When the check box is cleared, the position of each item in a wrapped list is determined by the chosen indentation level.

The status of the check box affects the formatting only if you have chosen to wrap lists, if the Do not wrap option is chosen, selecting or clearing the check box has no effect.

– Place ';' on new line
When the check box is cleared, the semicolon (`;`) is displayed after the last item in a chained call. If the check box is selected, the semicolon is moved to a new line:

```
$x = $x->one( "a", "b" )
                            ->two( "c", "d", "e" )
                            ->three( "fg" )
                            ->four
                    ;
```

Assignment statement

In this section, configure wrapping and subsequent alignment in assignment statements.

– Do not wrap - when this option is selected, no special wrapping style is applied.
With this option selected, the nested alignment and braces settings are ignored.
– Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.
– Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.
– Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
– Assignment sign on new line
– Align consecutive assignments
When this check box is selected, the assignment signs in consecutive assignment statements are aligned by the rightmost one:

```
$y            = foo( $x );
                    $intermediate = foo( $intermediate );
                    $result       = $result . $y . $intermediate;
```

When this check box is cleared, no alignment is applied:

```
$y = foo( $x );
                    $intermediate = foo( $intermediate );
                    $result = $result . $y . $intermediate;
```

Class field/constant groups

In this section, configure wrapping and subsequent alignment within lists of class properties (fields) or class constants .

– Align fields in columns
– Align constants

Array initializer

In this section, configure wrapping and subsequent alignment in array declarations.

- Do not wrap - when this option is selected, no special wrapping style is applied.
  With this option selected, the nested alignment and braces settings are ignored.
- Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.
- Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.
- Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
- Align when multiline
  When the check box is selected and the New line after '(' check box is cleared, each element, which after wrapping is positioned on a new line, is aligned by the first one, which remains unwrapped:

```
$colours = array( "blue",
                                      "red",
                                      "white",
                                      "green",
                                      "yellow" );
```

When both the Align when multiline and the New line after '(' check boxes are selected, all the elements ae aligned according to the indentation settings:

```
$colours = array(
                          "blue",
                          "red",
                          "white",
                          "green",
                          "yellow" );
```

Selecting or clearing the Align when multiline and the New line after '(' check boxes affects the formatting only if you have chosen to wrap lists, if the Do not wrap option is chosen, the status of the check boxes has no effect.

- New line after '('
  When this check box is selected, the first element of the array is displayed on a new line and all the elements of the array are aligned according to the indentation settings, regardless of the status of the Align when multiline check box.

  When this check box is cleared, the first element of the array remains on the same line, and the other elements are aligned according to the indentation settings.
- Place ')' on new line

Modifier list

- Wrap after modifier list
  When this check box is selected, the code is wrapped after a list of visibility modifiers:

```
protected
                      function Foo() {
                      }
```

When the check box is cleared, no wrapping is performed:

```
protected function Foo() {
                      }
```

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
Minimum Blank Lines	In the text boxes in this area, specify the number of blank lines to be present in the specified locations. **Warning!** These settings do not influence the number of blank lines before the first and after the last item.

PHPDoc

In this tab, configure the code style to be applied inside PHPDoc comments. Learn more about documenting PHP code at Creating PHP Documentation Comments .

Item	Description
Align parameter names	Select this check box to have the `&<paramname>` elements aligned.
Keep blank lines	Select this check box to suppress removing blank lines automatically.
Blank lines around parameters	Select this check box to have a blank line inserted above and below the section with `@param` tags.
Blank line before the first tag	Select this check box to have an blank line inserted above the first PHPDoc tag.
Align tag comments	Select this check box to have the `description` elements aligned.
Wrap long lines	Select this check box to have the text that exceeds the right margin wrapped to the next line.
Generated Doc Blocks	In this area, configure the code style to be applied within generated PHP documentation blocks, see Creating PHP Documentation Comments .

	– Use fully-qualified class names: select this check box to have IntelliJ IDEA specify a fully class names for properties, function parameters, `return` and `throws` values, etc.
`@throws` Tag Analysis	– Call tree analysis depth: specify the depth of analysis. IntelliJ IDEA can analyze exceptions up to 3 levels deep. – Ignore Runtime exceptions: If this check box is selected, the exceptions which can only be found at runtime (either based on `RuntimeException` or any of its subclasses), are ignored. – Ignore Logic exceptions: If this check box is selected, exceptions caused by the error in the program logic (either based on `LogicException` or any of its subclasses), are ignored.

Other

Item	Description
Indent code in PHP tags	Select this check box to have the code enclosed in `<?php>` tags indented against the opening `<?php` tag.
Convert True/False constants to upper case	Select this check box to have the `true` and `false` constants displayed in the upper case.
Convert Null constant to upper case	Select this check box to have the `null` constant displayed in the upper case.
Blank line before return statement	Select this check box to have IntelliJ IDEA automatically insert a blank line before each `return` statement.
Spaces around variables/expressions in brackets	– Select this check box to have IntelliJ IDEA insert spaces inside brackets during reformatting only if the brackets enclose a variable or an expression. This setting affects reformatting only if you have not configured force insertion of spaces inside brackets by selecting the Brackets check box under the Within node in the Spaces tab. This option helps you keep your code in accordance with the WordPress PHP Coding Standards . – If this check box is cleared, the spaces insertion policy depends on the settings under the Within node in the Spaces tab regardless of the type of content inside brackets: – if the Brackets check box is selected, spaces are always inserted – if the Brackets check box is cleared, spaces are never inserted.
Code Commenting	In this area, configure the code style options to be applied to comments. – Line comment at first column: - select this check box to have line comments start at the first column, without any indentation. Note that no extra blank spaces are added after the line comment characters. The check box is by default selected. When the check box is cleared, the line comments start from the minimum indentation within the selected code block to be commented.
Array declaration style	– Force short declaration style: select this check bo to have IntelliJ IDEA replace the `array()` constructs with `[]` in array declarations during reformatting, see Arrays. Syntax for details. – Align key-value pairs: select this check box to have the `=>` separators in key-value assignments aligned. – Add a comma after last element in multiline array: select this check box to have IntelliJ IDEA automatically insert a comma after the last item in declarations of multiline arrays to meet the required coding standard, for example, the Symfony coding standards .

Arrangement

In this tab, define a set of rules to re-arrange your PHP code according to your preferences.

Item	Description
Grouping Rules	Use this area to set the grouping rules. – Keep getters and setters together Select this check box to keep getter and setter methods together. By default, this check box is selected. – Keep overridden methods together Select this check box to group the overridden methods together by class and interface. In order: list, select keep or by name options. – Keep dependent methods together Select this check box to group the dependent methods together. In order: list, select depth-first or breadth-first options.
Matching rules	Use this area to define elements order as a list of rules, where every rule has a set of matches such as modifier or type. – - use this button to add a rule. The empty rule dialog window opens. – - use this button to remove the rule from the list. – - use this button to edit an existing rule. To see this button, navigate to the rule that you want to edit and click on the button. In pop-up window that opens, modify the rule fields. – - use these buttons to move the selected rule up or down.
Empty rule	Use this window to create a new matching rule or edit an existing one. You can select from the following filters: – Type - use this filter to choose classes or methods for your rule. Note that clicking twice on the type keyword will negate the condition. – Name - use this field to specify entry names in the rule. This filter matches only entry names such as field names, method names, class names and etc. The filter supports regular expressions and uses a standard syntax . The match is performed against the entire name. – Order - use this drop-down list to select the sorting order for the rule. This option is useful when more than one element uses the same matching rule. In this case, selecting Keep order will keep the same order as was set before the rearrangement and selecting Order by Name will sort the elements with the same matching rule by their names. – Aliases - this option displays aliases that were defined in the Rules Alias Definition dialog. You can remove the ones you do not need.
	This icon appears when you select Order by Name in Order option. The icon indicates that the items in this rule are alphabetized.

Code Style. Sass

File | Settings | Editor | Code Style | Sass for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | Sass for OS X

Use this page to configure formatting options for Sass files. View the result in the Preview pane on the right.

On this page:

– Tabs and Indents

– Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. SCSS

File | Settings | Editor | Code Style | SCSS for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | SCSS for OS X

Use this page to configure formatting options for SCSS files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. SQL

File | Settings | Editor | Code Style | SQL for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | SQL for OS X

Use this page to configure formatting options for SQL files. View the result in the Preview pane on the right.

On this page:

- General
- Tabs and Indents
- Spaces
- Wrapping and Braces
- Blank Lines
- Set from...

General

Use this tab to specify certain formatting behaviors in your code. The results are displayed in the Preview pane.

Item	Description
Word Case	In this node, specify whether the keywords, identifiers, or quoted identifiers case should change automatically. Click the right-hand column, and check the desired behavior on the menu to turn it on. The possible options are: – To upper: the elements are automatically converted to the upper case. – To lower: The elements are automatically converted to lower case. – Do not change: The case of elements is left as is.
Identifier quotations	Click the right-hand column, and check the desired behavior on the menu to turn it on. The possible options are: – Quote: the identifiers are automatically quoted. – Unquote: The identifiers are automatically unquoted. – Do not change: The identifiers are left as is.
New line before/after	Use these nodes to define how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of SQL element, select of clear the check box to the right. The results are highlighted in the Preview pane.
New line around semicolon	Use this check box to manage line delimiters around a semicolon.
Alignment	Select the check boxes next to the desired elements of source code to have them automatically aligned.

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and Braces

In this tab, customize the code style options, which IntelliJ IDEA will apply when reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Item	Description
Keep When Reformatting	Use the check boxes in this node to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line brakes* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the *Line brakes* check box before you reformat the source code .
Wrap inside	Use this node to define wrapping style for the lengthy elements. You can use different wrapping style options that are available from the drop-down list in the left-hand pane.
Values expression	Use this option to define wrapping style for values expressions. You can use different wrapping style options that are available from the drop-down list in the left-hand pane.
Wrapping Style Options	The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, expressions, or assignment statements). From the drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. Note If this option is selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.

Blank Lines

Use this tab to insert blank lines into your code.

Item	Description
Keep Maximum Blank Lines	Use this area to specify blank lines in your code. Use In code field to enter the number of lines you want to insert. The default number is 2.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. Stylus

File | Settings | Editor | Code Style | Stylus for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | Stylus for OS X

Use this page to configure formatting options for Stylus files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. TypeScript

File | Settings | Editor | Code Style | TypeScript for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | TypeScript for OS X

Use this page to configure formatting options for TypeScript files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Spaces
- Wrapping and braces
  - Right Margin (columns)
  - Wrap on typing
  - Keep when reformatting
  - Wrapping options
  - Alignment options
  - Braces placement options
- Blank lines
- Other
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.
Indent chained methods	Use this check box to configure indentation of chained method calls.

Spaces

Use this tab to specify where you want spaces in your code. To have IntelliJ IDEA automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which IntelliJ IDEA will apply on reformatting the source code . The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over indentation options .

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings .
- Yes - in this case IntelliJ IDEA uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that IntelliJ IDEA will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you reformat the source code .

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

Item	Description
Wrapping style	From this drop-down list, select the desired wrapping style: – Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.

Alignment options

Item	Description
Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
Align parenthesised when multiline	If this check box is selected, a parenthesised code construct starts at the column of the opening parenthesis each next line. Otherwise, the position of a code construct is determined by the current indentation level.
`<character(s)>` on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after `<character>`	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, `else if` statements are located in the same line. Otherwise, `else if` statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the `case` statement is located at the corresponding indent level. Otherwise, `case` statement is placed at the same indent level with `switch`.

Braces placement options

Item	Description
Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: – End of line - select this option to have the opening brace placed at the declaration line end. – Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line. – Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line. – Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level. – Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.

Force braces	From this drop-down list, choose the braces introduction method for `if`, `for`, `while`, and `do () while` statements. The available options are:
	– Do not force - select this option to suppress introducing braces automatically.
	– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that IntelliJ IDEA analyzes the number of lines in the entire statement but not only its condition. For example, if the code is
	`<?php`
	`if (true)`
	`return false;`
	IntelliJ IDEA will insert braces automatically:
	`<?php`
	`if (true)`
	`{`
	`return false;`
	`}`
	– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want IntelliJ IDEA to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

Item	Description
Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Other

Item	Description
Align object properties	From the drop-down list, select the type of objects' alignment: – Do not align: the attributes in sequential lines will be not aligned. – On colon: the attributes in sequential lines will be aligned against the colon. – On value: the attributes in sequential lines will be aligned against the value.

Use semicolon to terminate statements	Select this check box to have statements terminated with a semicolon.
Generated Code	In this area, configure the code style in `import` statements, see Creating Imports , section Importing TypeScript Symbols. – String literal type: from this drop-down list, choose the type of quotes to be used in `import` statements. The available options are Single quotes and Double quotes.
Use 'public' modifier	Use this check box to have the public access modifier inserted or omitted in the generated code. For example, during generation of a public method from the following:

```
class Test {
                    public test():void {
                    var x = 1;
                    }
                    }
```

– If the check box is selected, the public access modifier is automatically inserted in the generated code:

```
class Test {
                     public test():void {
                     this.extracted();
                     }
                     extracted(){
                     var x = 1;
                     }
                     }
```

– If the check box is cleared, the public access modifier is omitted during code generation:

```
class Test {
                     public test():void {
                     this.extracted();
                     }
                     public extracted(){
                     var x = 1;
                     }
                     }
```

See TypeScript Language Handbook , chapter Private/Public Modifiers .

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. Velocity

File | Settings | Editor | Code Style | Velocity for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | Velocity for OS X

Use this page to configure formatting options for Velocity files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. XML

File | Settings | Editor | Code Style | XML for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | XML for OS X

Use this page to configure formatting options for XML files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Other
- Arrangement
- Android
- Set from...

Tabs and Indents

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the `Tab` key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	If this check box is cleared, then IntelliJ IDEA will keep indents on the empty lines as if they have contained some code. If this check box is not selected, IntelliJ IDEA will delete the tabulation characters.

Other

Item	Description
Right Margin	Use these settings to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings will be used.
Wrap on typing	Use these settings settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options: – Default - in this case IntelliJ IDEA uses the Wrap on typing option that is specified in the global settings . – Yes - in this case the value in the specified right margin is used. – No - in this case this option is switched off and a line can exceed the number that is specified in the right margin.
Keep line breaks	Select this check box to have IntelliJ IDEA honor line breaks when reviewing XML files in the editor.
Keep line breaks in text	Select this check box to have IntelliJ IDEA honor line breaks in attributes (for example, lengthy descriptions) when reviewing XML files in the editor.
Keep blank lines	In this text box, specify the minimum number of sequential blank lines to be retained after reformatting.
Wrap attributes	Use this drop-down list to determine how attribute lines should be wrapped. The available options are: – Do not wrap - if this option is selected, no special wrapping style is applied to the code. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped to give one element per line with proper indentation. – Wrap always - select this option to have all elements in lists wrapped to give one element per line with proper indentation.
Wrap text	Select this check box to have long lines wrapped according to the code style settings.
Align attributes	Select this check box to have attributes in sequential lines aligned.
Keep white spaces	When this check box is selected, the editor preserves all whitespaces within tags. The same refers also to the indents, and line breaks.
Spaces	In this area, define the usage of spaces for attributes and tag names. – Around "=" in attribute: select this check box to have spaces added around the "=" symbol in attributes. – After tag name: select this check box to have spaces added after tag names. – In empty tag: select this check box to have spaces added in empty tags.
CDATA	In this area, define the usage of whitespaces around and inside CDATA sections in MXML files: – Whitespaces around: from the drop-down list, choose how whitespaces around `CDATA` will be treated. – Preserve: all whitespaces will be left intact after reformatting. – Remove (keep with tags): all whitespaces around `CDATA` will be removed, and tags will be kept on the same lines. – New lines: new lines will be added before and after `CDATA` . – Keep whitespaces inside: If this check box is selected, whitespaces will be preserved after `CDATA[` and before `]]` .

Arrangement

This tab lets you define a set of rules that rearranges your code according to your preferences.

Item	Description
Matching rules	Use this area to define elements order as a list of rules, where every rule has a set of matches such as modifier or type. – - use this button to add a rule. The empty rule dialog window opens. – - use this button to remove the rule from the list. – - use this button to edit an existing rule. To see this button, navigate to the rule that you want to edit and click on the button. In pop-up window that opens, modify the rule fields. – - use these buttons to move the selected rule up or down.
Empty rule	Use this window to create a new matching rule or edit an existing one. You can select from the following filters: – Type - use this filter to choose classes or methods for your rule. Note that clicking twice on the type keyword will negate the condition. – Name - use this field to specify entry names in the rule. This filter matches only entry names such as field names, method names, class names and etc. The filter supports regular expressions and uses a standard syntax . The match is performed against the entire name. – Namespace - use this field to specify the namespace in the rule. It lets you specify a rule that controls a namespace attribute position. – Order - use this drop-down list to select the sorting order for the rule. This option is useful when more than one element uses the same matching rule. In this case, selecting Keep order will keep the same order as was set before the rearrangement and selecting Order by Name will sort the elements with the same matching rule by their names. – Aliases - this option displays aliases that were defined in the Rules Alias Definition dialog. You can remove the ones you do not need.
	This icon appears when you select Order by Name in Order option. The icon indicates that the items in this rule are alphabetized.
Additional Settings	Use this area to set additional arrangement options. The Force rearrange drop-down list lets you select options that affect the Rearrange entries check box in the Reformat Code dialog. You can select from the following options: – Use current mode (toggled in the Reformat Code dialog) - In this case the Rearrange entries check box stays active and you can modify it in the Reformat Code dialog. – Always - In this case the Rearrange entries check box is selected and becomes read-only. – Never - In this case the Rearrange entries check box is cleared and becomes read-only.

Android

Item	Description
Use custom formatting settings for Android XML files	Use this check box to set a custom formatting for the Android XML files. This might be helpful if you need to format Android files differently from other XML files or need to use specific Android formatting options. If this check box is not selected, the default XML files formatting is applied.
AndroidManifest.xml	Use this area to set the following formatting for AndroidManifest.xml: – Wrap attributes - use this drop-down list to determine how attribute lines should be wrapped. See the available options . – Insert line break before first attribute - select this check box to insert a line before the first attribute. – Insert line break after last attribute - select this check box to insert a line after the last attribute. – Group tags with the same name - select this check box to group tags with the same name.
Layout Files	Use this area to set the following formatting for the layout files: – Wrap attributes - use this drop-down list to determine how the attribute lines should be wrapped. See the available options . – Insert line break before first attribute - select this check box to insert a line break before the first attribute. – Insert line break after last attribute - select this check box to insert a line after the last attribute. – Insert blank line before tag - select this check box to insert a blank line before the tag.
Value Resources Files and Selectors	Use this area to set the following formatting for value resource files and selectors: – Wrap attributes - use this drop-down list to determine how attribute lines should be wrapped. See the available options . – Insert line breaks around style declaration - select this check box to insert line breaks around a style declaration.
Other XML resource files	Use this area to set Wrap attributes for other XML resource files. See the available options . – Insert line break before first attribute - select this check box to insert line breaks before the first attribute. This option is selected by default. – Insert line break after last attribute - select this check box to insert line breaks after the last attribute.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Code Style. YAML

File | Settings | Editor | Code Style | YAML for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Code Style | YAML for OS X

Use this page to configure formatting options for YAML files. View the result in the Preview pane on the right.

On this page:

- Tabs and Indents
- Set from...

Tabs and Indents

Item	Description

Use tab character	- If this check box is selected, tab characters are used:
	  - On pressing the Tab key
	  - For indentation
	  - For code reformatting
	- When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.

Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Other File Types

File | Settings | Code Style | Other File Types for Windows and Linux

IntelliJ IDEA | Preferences | Code Style | Other File Types for OS X

Ctrl+Alt+S

On this page:

- Scheme
- Text files and unsupported file types

Scheme

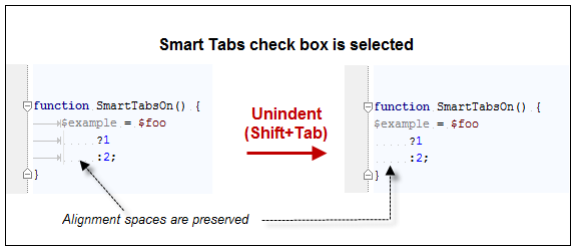
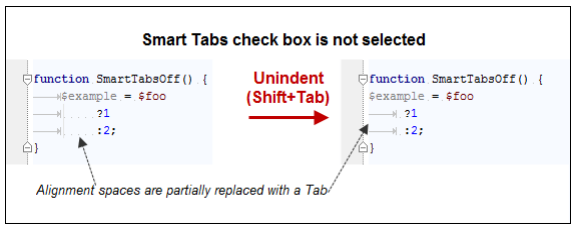
In this area, choose the code style scheme and change it as required. Settings of a code style scheme are automatically applied every time IntelliJ IDEA generates, refactors, or reformats your code.

The code styles are defined at the project level and at the global level.

- At the Project level, settings are grouped under the Project scheme, which is predefined and is marked in bold font. The Project style scheme applies to the current project only.
  The default scheme cannot be changed! When any modifications are made, IntelliJ IDEA automatically creates a copy of the default scheme. If you want to use a different project level scheme, choose the desired scheme and copy its settings to project.
- At the Global level, settings are grouped under the predefined Default scheme, and any other scheme created by the user. The global settings are used when the user doesn't want to keep code style settings with the project and share them.

Item	Description
Scheme	From this drop-down list, select the scheme to be used. The predefined schemes are shown in bold. The custom schemes, ones created as copies of the predefined schemes, are in plain text.
Manage	Click this button to open the Code Style Schemes dialog that lets you manage the schemes.
Reset	This link appears in the upper-right corner of the dialog when the current style or scheme are modified. Click this button to discard changes.

Text files and unsupported file types

Item	Description
Use tab character	– If this check box is selected, tab characters are used: – On pressing the Tab key – For indentation – For code reformatting – When the check box is cleared, IntelliJ IDEA uses spaces instead of tabs.
Smart tabs	– If this check box is selected, IntelliJ IDEA inserts tabs for indentation and reformatting. IntelliJ IDEA inserts tabs in the beginning of each line until a symbol other than the tab is inserted. After that when you press the tab key, IntelliJ IDEA will enter spaces.  – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment.  The Smart Tabs check box is available if the Use Tab Character check box is selected.
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Inspections

File | Settings | Editor | Inspections for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Inspections for OS X

`Ctrl+Alt+S`

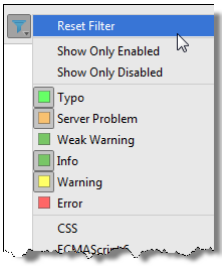
Use this page to customize inspection profiles , configure inspection severities , disable and enable inspections , and configure inspections for different scopes . The page is divided into the following areas:

- Profile management
- Toolbar
- Inspection severity and scopes
- Options

Profile management

Item	Description
Profile	From this drop-down list, select the name of the profile to configure. All modified inspections are highlighted.
Manage	Click this drop-down list to reveal the submenu of check commands:
Share to team members	Select this check command to make the selected profile available for your team, i.e to make it the project-level profile. The settings of the profile are passed to XML file that is located in the `<project>/.idea/inspectionProfiles` directory. The profile then can be shared between your team members or you can commit the profile to VCS. If you create a profile for your own purposes, make sure that this check box is cleared.
Copy	Choose this command to create a copy, based on the current profile .
Rename	Choose this command to change the name of the current profile in the Profile field.
Delete	Choose this command to delete the current profile. The pre-defined profiles cannot be deleted, so this command is only available for the user-defined profiles.
Export	Choose this command to export the selected profile as an `xml` file.
Import	Choose this command to import a profile from an `xml` file.

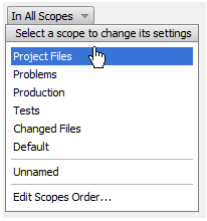
Toolbar

Item	Tooltip and Shortcut	Description
		Use this text box to search through the list of inspections. In the search field, start typing the desired inspection name, or any characters contained in the inspection name or description. IntelliJ IDEA shows the list of matching occurrences. As you type a search string, the matching inspections are highlighted. To finalize the search, press `Enter` . The used search strings are memorized in the history list. – : Click this button to reveal the history list. – : Click this button to clear the search history.
	Filter Inspections	Click this button to show the list of available filters:  Click the desired filters to reduce the list. The command Reset Filters becomes available, if some of the filters are checked.

	Expand All/Collapse All
	`Ctrl+NumPad Plus`
	`Ctrl+NumPad -`
		Click these buttons to have all inspection nodes expanded/collapsed.

	Reset to Empty	Click this button to have all the check boxes of the profile cleared and thus disable all the profile inspections.
	Advanced Settings	Click this button the show the menu with the following check commands: – Disable new inspections by default: select this check box, if you don't want the new inspections that appear after IntelliJ IDEA update, to become available. – Reset to Default settings: select this check box to discard all changes.

Inspection severity and scopes

Item	Description
Description	This read-only field shows the description of the selected inspection.
Inspection severity	From this drop-down list, select the desired severity to assign to the current inspection. The possible options are: – Typo – Server problem – Info – Weak warning – Warning – Error
Scopes	Click this drop-down list to reveal the list of available scopes :  Clicking a scope in the list results in showing the scopes toolbar: `Alt+Insert` — Click this button to specify the scope for the selected inspection. `Alt+Delete` — Click this button to delete the selected scope for the current inspection. Choosing the option Edit Scopes Order results in showing the Scopes Order dialog box, where one can change the order of scopes using the up and down arrows ( ) or keyboard shortcuts (`Alt+Up` / `Alt+Down`). It is possible to select several inspections and add/remove scopes for the entire selection.

Options

This area is only available for some types of inspections, provided that an inspection of this type is enabled (the check box next to it is selected). Use the controls in this area to re-configure the default inspection settings.

File and Code Templates

File | Settings | Editor | File and Code Templates for Windows and Linux

IntelliJ IDEA | Preferences | Editor | File and Code Templates for OS X

`Ctrl+Alt+S`

Files can be created according to pre-defined templates (file templates). Use this page to view, edit, and create such templates.

Different groups of templates are located on different tabs .

When you select a template, its contents and description are displayed in the right-hand part of the page.

- Per-project vs default scheme
- Tabs
- Toolbar
- Template settings and contents

Per-project vs default scheme

Item	Description
Scheme	From this drop-down list, choose whether file and code template settings pertain to the entire workspace, or the current project: – Default scheme is selected, when file and code templates are global. – Project scheme is selected, if you want to use the sharable project-specific file and code templates.

Refer to the section Project and IDE Settings for details.

Tabs

Tab	Description
Files	This tab displays the available file templates. You can edit the existing templates, or create new ones. Note The templates shown in bold font cannot be deleted; their names and extensions cannot be edited.
Includes	This tab shows the templates for reusable fragments that can be included in file templates. You can edit the existing templates, or create new ones.
Code	This tab displays built-in snippets, i.e. templates for code fragments that IntelliJ IDEA can generate in various typical situations , for example, for generating implemented or overridden method bodies . You can edit the existing snippets, but you cannot create new ones.
Other	This tab displays groups of templates related to various application servers and frameworks. You can edit the existing templates, but you cannot create new ones.

Toolbar

Item	Tooltip	Description
+	Create Template	Click this button to create a new template in the currently opened tab. This option is only available in the Files and the Includes tabs. The location of the new template is defined by the Schema drop-down list. If
–	Remove Template	Click this button to delete the selected template. This option is only available for certain templates in the Files and the Includes tabs.
	Copy Template	Click this button to create a copy of the selected template. This option is only available in the Files and the Includes tabs.
	Reset to Default	Click this button to revert the selected template to its original state. This option is only available for templates that have been modified (they are highlighted in blue).
N/A	Reset	This link appears in the top-right corner of the page when you start editing a template. Clicking this link resets all unsaved changes to any template in any tab.

Template settings and contents

Item	Description
Name	This text box appears when a new template is created. Specify the name of the new template.
Extension	In this text box, specify the extension. IntelliJ IDEA will apply this template when new files of this type are created.
Reformat according to style	Select this check box, to have IntelliJ IDEA reformat generated stub files according to the style defined on the Code Style page . This option is only available in the Files tab.
Enable Live Templates	Select this check box to use a live template inside a file template. So doing, one has to put the live template fragments into Velocity escape syntax. For example: `#[[ $MY_VARIABLE$ $END$ ]]#` Thus, one can specify the cursor position. Note that it is required to use the live template variables here!

Template text

Edit the template contents. You can use:

- Plain text.
- `#parse` directives to work with template includes .
- Custom variables. Their names can be defined right in the template through the `#set` directive or will be defined during the file creation.
- Variables to be expanded into corresponding values in the `${<variable_name>}` format.

  The available predefined file template variables are:

  - `${PACKAGE_NAME}` - the name of the target package where the new class or interface will be created.
  - `${PROJECT_NAME}` - the name of the current project.
  - `${FILE_NAME}` - the name of the PHP file that will be created.
  - `${NAME}` - the name of the new file which you specify in the New File dialog box during the file creation.
  - `${USER}` - the login name of the current user.
  - `${DATE}` - the current system date.
  - `${TIME}` - the current system time.
  - `${YEAR}` - the current year.
  - `${MONTH}` - the current month.
  - `${DAY}` - the current day of the month.
  - `${HOUR}` - the current hour.
  - `${MINUTE}` - the current minute.
  - `${PRODUCT_NAME}` - the name of the IDE in which the file will be created.
  - `${MONTH_NAME_SHORT}` - the first 3 letters of the month name. Example: Jan, Feb, etc.
  - `${MONTH_NAME_FULL}` - full name of a month. Example: January, February, etc.

  IntelliJ IDEA provides a set of additional variables for PHP include templates , that is, templates of reusable fragments that can be included in other PHP file template. The built-in PHP include templates are intended for generating file headers and PHPDoc documentation comments . The following variables are available in PHP include templates:

  - `${NAME}` - the name of the class, field, or function (method) for which the PHPDoc comment will be generated.
  - `${NAMESPACE}` - the fully qualified name (without a leading slash) of the class or field namespace.
  - `${CLASS_NAME}` - the name of the class where the field to generate the PHPDoc comment for is defined.
  - `${STATIC}` - gets the value `static` if the function (method) or field to generate the comment for is `static` . Otherwise evaluates to an `empty string` .
  - `${TYPE_HINT}` - a prompt for the `return` value of the function (method) to generate the comment for. If the return type cannot be detected through the static analysis of the function (method), evaluates to `void` .
  - `${PARAM_DOC}` - - a documentation comment for parameters. Evaluates to a set of lines `@param type name` . If the function to generate comments for does not contain any parameters, the variable evaluates to empty content.
  - `${THROWS_DOC}` - a documentation comment for exceptions. Evaluates to a set of lines `@throws type` . If the function to generate comments for does not throw any exceptions, the variable evaluates to empty content.
  - `${DS}` - a dollar character (`$`). The variable evaluates to a plain dollar character (`$`) and is used when you need to escape this symbol so it is not treated as a prefix of a variable.
  - `${CARET}` - indicated the position of the caret after generating and adding the comment.

    This `${CARET}` variable is applied only when a PHPDoc comment is generated and inserted during file creation. When a PHPDoc comment is created through Code | Generate | PHPDoc block, multiple selection of functions or methods is available so documentation comments can be created to several classes, functions, methods, or fields. As a result, IntelliJ IDEA cannot "choose" the block to apply the `${CARET}` variable in, therefore in this case the `${CARET}` variable is ignored.

  Treating dollar sign

  - You can prevent treating dollar characters (`$`) in template variables as prefixes. If you need a dollar character (`$` inserted as is, use the `${DS}` file template variable instead. When the template is applied, this variable evaluates to a plain dollar character (`$`).

    Examples:

    - To use some version control keywords (such as `$Revision$` , `$Date$` , etc.) in your default class template, write `${DS}` instead of the dollar prefix (`$`).
    - The template code `${DS}this` will be rendered as `$this` .

  Note IntelliJ IDEA doesn't prompt for the values of Velocity variables defined with `#set` .

Description

This read-only field provides information about the template, its predefined variables, and the way they work. This field is not available in custom templates.

File Encodings

File | Settings | Editor | File Encodings for Windows and Linux

IntelliJ IDEA | Preferences | Editor | File Encodings for OS X

`Ctrl+Alt+S`

Use this dialog to configure encoding options for a project and for the entire IDE.

The file or directory encodings take precedence over the project encoding , which, in turn, takes precedence over the IDE encoding .

If the file or directory encodings are not defined, then the project encoding is taken. If the project encoding cannot be taken (for example, if a project is not yet created), then the IDE encoding is taken.

Item	Description
IDE Encoding	From this drop-down list, choose the encoding to be used when no project is currently opened. The encoding will be applied, for example, when you specify settings of a default project or check out sources from a version control storage. Choose System Default to have the default encoding of your operating system used or choose a specific encoding.
Project Encoding	From this drop-down box, choose the default encoding to use in the folders for which no encoding is appointed in the Default encoding field below. Choose System Default to have the default encoding of your operating system used or choose a specific encoding.
File/Directory	This column displays the project tree view.
Default encoding	This column displays encoding for a file or directory, if applicable. If encoding is defined within a file, it cannot be configured, and is shown in grey font. If encoding is configurable, click the Default Encoding column for a selected file or directory, and choose encoding from the drop-down list. Encoding information embedded in a file overrides the selected one; encoding information for the nested files or directories overrides that for the outer directories or the whole project.
Default encoding for properties files	Use this drop-down list to define encoding for the properties files in the project. According to the Java API, the `load(InputStream) / store(OutputStream, String)` methods of the `java.util.Properties` class, use ISO 8859-1 encoding for input/output stream. It is advisable to use this encoding unless you have special reasons to change it. In this case, you can select the desired encoding from the drop-down list; in particular, you can use encoding defined for the whole project.

Transparent native-to-ascii conversion	Select this option to show in properties files the national characters (non-ISO 8859-1), stored as escape sequences.
	For more details refer to Configuring Encoding for Properties Files .
	
	If this check box is not selected, the national characters will not be shown.
	Compare two representations of the national characters:
	
	test-i18n_de.properties ×
	`key1=Was ich nicht weiß, macht mich nicht heiß`
	`key2=value2_de`
	`test.value=test value`
	`test.key=test value`
	
	test-i18n_de.properties ×
	`key1=Was ich nicht wei\u00df, macht mich nicht hei\u00df`
	`key2=value2_de`
	`test.value=test value`
	`test.key=test value`

Live Templates

File | Settings | Editor | Live Templates for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Live Templates for OS X

`Ctrl+Alt+S`

Use this page to create, manage, and edit live templates .

On this page:

– List of available live templates
– Context menu of a live template
– Template editing area
– Side note about pe-defined template variables
– Predefined functions to be used in live template variables

List of available live templates

Item	Tooltip and shortcut	Description
By default expand with		Use this drop-down list to specify the default invocation key for all templates. Individual expansion keys for the particular templates are defined in the editing area . If the standard expansion keys (Tab, Enter, or Space) are not desirable, select the Custom option from this drop-down list. When Custom is selected, the Change link appears next to the drop-down, leading you to the Keymap page.
Live Templates		This list shows all currently available template abbreviations supplied with their descriptions. The abbreviations are grouped below nodes and sorted alphabetically within each group. To activate a template or an entire group, select the check box to its left. **Note** – Only active templates are displayed upon invoking live templates in the editor. – If a template is active, the editor is sensitive to its abbreviation. Otherwise, the abbreviation is considered merely a set of characters.
+	Add `Alt+Insert`	Click this button to have a new template item added to the current group of template. You can define the template abbreviation, description, text, variables, expansion key, and context in the editing area below .
–	Remove `Delete`	Click this button to have the selected live template removed from the list.
	Duplicate	Click this button to create a new template based on the selected template. A new template item is added to the current node and the fields in the Template Text area show the definition of the selected template.
	Restore	Click this button to restore the deleted live templates. This button is only enabled when the changes are applied.

Context menu of a live template

Item	Description
Move	Choose a group to move the selected template to.
Change context	Choose this command to modify the set of contexts where the current template is enabled. Upon choosing this command, a list of supported language contexts is displayed. To make IntelliJ IDEA consider a context sensitive to the template, select a check box next to the context name. The available context types depend on the enabled plugins.
Copy	Choose this command to create a serialized template XML in the system clipboard. Refer to section Sharing Live Templates .
Paste	Choose this command to paste an XML representation of the copied templates to the selected group of templates . Refer to section Sharing Live Templates .
Restore defaults	This command only appears on the context menus of the modified templates, marked blue. Choose this command to restore the default template settings.

Template editing area

The focus is moved to this area in the following cases:

– When you click the Add + or Copy button.
– When you select a live template in the list.
– When you select a fragment of code in the editor and choose Tools | Save as Live Template .

Use controls of this area to create new live templates and edit the settings for the existing ones.

You can navigate through the Template Text Area using the hot keys that are marked in the field labels.

Item	Description

Abbreviation	In this text box, specify the template abbreviation .
Description	In this text box, provide optional description of a template or an example of its usage.
Template Text	In this text box, type the template body that may contain plain text and variables in the format `$<variable name>$` . When editing the live template variables, mind the following helpful hints: – If you need a dollar sign (`$`) in the template text, escape it by duplicating this character (`$$`). – To change the variables in a template, click the Edit Variables button and configure the variables as described in Creating and Editing Template Variables . The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from `$END$` or `$SELECTION$` .

Side note about pe-defined template variables

IntelliJ IDEA supports two predefined live template variables: `$END$` and `$SELECTION$` .

You cannot edit the predefined live template variables `$END$` and `$SELECTION$` .

– `$END$` indicates the position of the cursor after the template is expanded. For example, the template `return $END$;` will be expanded into

```
return ;
```

with the cursor positioned right before the semicolon.

– `$SELECTION$` is used in surround templates and stands for the code fragment to be wrapped. After the template is expanded, the selected text is wrapped as specified in the template.
For example, if you select `EXAMPLE` in your code and invoke the `"$SELECTION$"` template via the assigned abbreviation or by pressing Ctrl+Alt+T and selecting the desired template from the list, IntelliJ IDEA will wrap the selection in double quotes as follows:

```
"EXAMPLE"
```

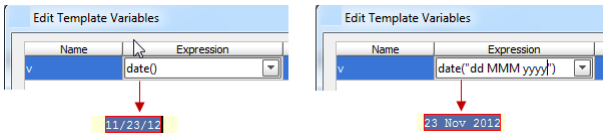
.

Applicable in:	This read-only field shows the languages and/or pieces of code where the editor should be sensitive to the template. Upon pressing Ctrl+J in such context, IntelliJ IDEA displays a list of templates that are valid for this context.
Change	Click this link to modify the set of contexts where the current template is enabled. Upon clicking the link, a list of supported language contexts is displayed. To make IntelliJ IDEA consider a context sensitive to the template, select a check box next to the context name. The available context types depend on the enabled plugins.
Edit Variables	Click this button to open the Edit Template Variables dialog box, where you can define how IntelliJ IDEA should process template variables upon template expansion. The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from `$END$` or `$SELECTION$` . The Edit Template Variables dialog box contains a complete list of available functions. See the list of predefined functions below on this page.
Options	In this area, define the behavior of the editor when a template is expanded. – Expand with - from this drop-down list, choose the key to invoke the template. – Reformat according to style - select this check box to have IntelliJ IDEA automatically reformat the expanded text according to the current style settings, defined on the Code Style page . – Use static import if possible - select this check box to have IntelliJ IDEA add static import statements instead of inserting `Class.methodName()` . – Shorten FQ names - select this check box to have IntelliJ IDEA truncate fully qualified names in the expanded template and add the corresponding import statements.

Predefined functions to be used in live template variables

Warning! Note that the list of predefined functions in IntelliJ IDEA depends upon the installed and enabled plugins.

Item	Description
`annotated("annotation qname")`	Creates a symbol of type with an annotation that resides at the specified location. For an example, see Live Templates in the iterations group.
`arrayVariable()`	Suggests all array variables applicable in the current scope. For an example, see Live Templates in the iterations group.
`anonymousSuper()`	Suggests a supertype for a Kotlin object expression.
`camelCase(String)`	Returns the string passed as a parameter, converted to camel case. For example, `my-text-file` / `my text file` / `my_text_file` will be converted to `myTextFile` .
`capitalize(String)`	Capitalizes the first letter of the name passed as a parameter.
`capitalizeAndUnderscore(sCamelCaseName)`	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is `FooBar` , then the function returns `FOO_BAR` .
`castToLeftSideType()`	Casts the right-side expression to the left-side expression type. It is used in the iterations group to have a single template for generating both raw-type and Generics Collections.
`className(sClassName)`	Returns the name of the current class (the class where the template is expanded).
`classNameComplete()`	This expression substitutes for the class name completion at the variable position.
`clipboard()`	Returns the contents of the system clipboard.
`camelCase(String)`	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is `foo_bar` , then the function returns `fooBar` .
`complete()`	This expression substitutes for the code completion invocation at the variable position.
`completeSmart()`	This expression substitutes for the smart type completion invocation at the variable position.
`componentTypeOf (<array variable or array type>)`	Returns component type of an array. For example, see the Live Templates in the iterations group in the other group.

`currentPackage()`	Returns the current package name.
`date(sDate)`	Returns the current system date in the specified format. By default, the current date is returned in the default system format. However, if you specify date format in double quotes, the date will be presented in this format: 
`decapitalize(sName)`	Replaces the first letter of the name passed as a parameter with the corresponding lowercase letter.
`descendantClassEnum(<String>)`	Shows the children of the class entered as a string parameter.
`enum(sCompletionString1,sCompletionString2,...)`	List of comma-delimited strings suggested for completion at the template invocation.
`escapeString(sEscapeString)`	Escapes the specified string.
`expectedType()`	Returns the type which is expected as a result of the whole template. Makes sense if the template is expanded in the right part of an assignment, after return, etc.
`fileName(sFileName)`	Returns file name with extension.
`fileNameWithoutExtension()`	Returns file name without extension.
`firstWord(sFirstWord)`	Returns the first word of the string passed as a parameter.
`groovyScript("groovy code")`	Returns Groovy script with the specified code. You can use `groovyScript` macro with multiple arguments. The first argument is a script text that is executed or a path to the file that contains a script. The next arguments are bound to `_1, _2, _3, ..._n` variables that are available inside your script. Also, `_editor` variable is available inside the script. This variable is bound to the current editor.
`guessElementType (<container>)`	Makes a guess on the type of elements stored in a `java.util.Collection`. To make a guess, IntelliJ IDEA tries to find the places where the elements were added to or extracted from the container.
`iterableComponentType(<ArrayOrIterable>)`	Returns the type of an iterable component, such as an array or a collection.
`iterableVariable()`	Returns the name of a variable that can be iterated.
`lineNumber()`	Returns the current line number.
`lowercaseAndDash(String)`	Returns lower case separated by dashes, of the string passed as a parameter. For example, the string `MyExampleName` is converted to `my-example-name`.
`methodName()`	Returns the name of the embracing method (where the template is expanded).
`methodParameters()`	Returns the list of parameters of the embracing method (where the template is expanded).
`methodReturnType()`	Returns the type of the value returned by the current method (the method within which the template is expanded).
`qualifiedClassName()`	Returns the fully qualified name of the current class (the class where the template is expanded). Tip Clear the Shorten FQ names check box.
`rightSideType()`	Declares the left-side variable with a type of the right-side expression. It is used in the iterations group to have a single template for generating both raw-type and Generics Collections.
`snakeCase(sCamelCaseText)`	Returns snake_case string out of CamelCase string passed as a parameter.
`spaceSeparated(String)`	Returns string separated with spaces out of CamelCase string passed as a parameter. For example, if the string passed as a parameter is `fooBar`, then the function returns `foo bar`.
`subtypes(sType)`	Returns the subtypes of the type passed as a parameter.
`suggestIndexName()`	Suggests the name of an index variable. Returns `i` if there is no such variable in scope, otherwise returns `j` if there is no such variable in scope, etc.
`suggestVariableName()`	Suggests the name for a variable based on the variable type and its initializer expression, according to your code style settings that refer to the variable naming rules. For example, if it is a variable that holds an element within iteration, IntelliJ IDEA makes a guess on the most reasonable names, also taking into account the name of the container being iterated.
`suggestFirstVariableName(sFirstVariableName)`	Doesn't suggest `true, false, this, super`.
`time(sSystemTime)`	Returns the current system time.
`typeOfVariable(VAR)`	Returns the type of the variable passed as a parameter.
`underscoresToCamelCase(sCamelCaseText)`	Returns the string passed as a parameter with CamelHump letters substituting for underscores. For example, if the string passed as a parameter is `foo_bar`, then the function returns `fooBar`.
`underscoresToSpaces(sParameterWithSpaces)`	Returns the string passed as a parameter with spaces substituting for underscores.
`user()`	Returns the name of the current user.
`variableOfType(<type>)`	Suggests all variables that may be assigned to the type passed as a parameter, for example variableOfType("java.util.Vector"). If you pass an empty string ("") as a parameter, suggests all variables regardless of their types.
`JsArrayVariable`	Returns JavaScript array name.
`jsClassName()`	Returns the name of the current JavaScript class.
`jsComponentType`	Returns the JavaScript component type.
`jsMethodName()`	Returns the name of the current JavaScript method.
`jsQualifiedClassName`	Returns the complete name of the current JavaScript class.

`jsSuggestIndexName`	Returns a suggested name for an index.
`jsSuggestVariableName`	Returns a suggested name for a variable.

Edit Template Variables Dialog

File | Settings | Live Templates - Edit Variables for Windows and Linux

IntelliJ IDEA | Preferences | Live Templates - Edit Variables for OS X

`Ctrl+Alt+S`

The dialog opens when you click the Edit Variables button in the Template Text area on the Live Templates page.

The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from `$END$` or `$SELECTION$`.

Use this dialog box to create and edit expressions for variables in the selected live template.

On this page:

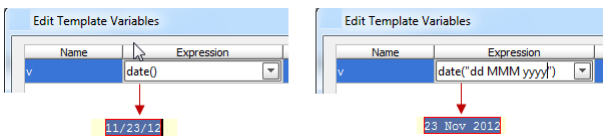
– Controls

– Predefined Functions to Use in Live Template Variables

Controls

Item	Description
Name	In this text box, view or edit the variable name in the format `$<variable_name>$`.
Expression	In this text box, specify the expression to have the value of the corresponding template input field calculated automatically. This expression may contain constructs of the following basic types: – String constants in double quotes. – The name of another variable defined in a live template. – Predefined functions with possible arguments. Type an expression manually or select a predefined function from the drop-down list. The list shows also the number and type of parameters, if any, for the selected function. The available functions are listed alphabetically in the Functions table.
Default value	In this text box, specify the default string to be entered in the corresponding input field of the expanded template, if the expression does not give any result after calculation. Note that a default value of a variable is an expression that can refer to other live template variables. To define the default value as a literal, enclose it in quotation marks.
Skip if defined	Select this check box to have IntelliJ IDEA proceed with the next input field, if the value of the current input field is defined.
Move Up / Move Down	Use these buttons to change the order of variables in the list. The order of variables in the table determines the order in which IntelliJ IDEA will switch between the corresponding input fields when the template is expanded.

Predefined Functions to Use in Live Template Variables

Item	Description
`annotated("annotation qname")`	Creates a symbol of type with an annotation that resides at the specified location. For an example, see Live Templates in the iterations group.
`arrayVariable()`	Suggests all array variables applicable in the current scope. For an example, see Live Templates in the iterations group.
`anonymousSuper()`	Suggests a supertype for a Kotlin object expression.
`camelCase(String)`	Returns the string passed as a parameter, converted to camel case. For example, `my-text-file` / `my text file` / `my_text_file` will be converted to `myTextFile`.
`capitalize(String)`	Capitalizes the first letter of the name passed as a parameter.
`capitalizeAndUnderscore(sCamelCaseName)`	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is `FooBar`, then the function returns `FOO_BAR`.
`castToLeftSideType()`	Casts the right-side expression to the left-side expression type. It is used in the iterations group to have a single template for generating both raw-type and Generics Collections.
`className(sClassName)`	Returns the name of the current class (the class where the template is expanded).
`classNameComplete()`	This expression substitutes for the class name completion at the variable position.
`clipboard()`	Returns the contents of the system clipboard.
`camelCase(String)`	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is `foo_bar`, then the function returns `fooBar`.
`complete()`	This expression substitutes for the code completion invocation at the variable position.
`completeSmart()`	This expression substitutes for the smart type completion invocation at the variable position.
`componentTypeOf (<array variable or array type>)`	Returns component type of an array. For example, see the Live Templates in the iterations group in the other group.
`currentPackage()`	Returns the current package name.
`date(sDate)`	Returns the current system date in the specified format. By default, the current date is returned in the default system format. However, if you specify date format in double quotes, the date will be presented in this format: 

`decapitalize(sName)`	Replaces the first letter of the name passed as a parameter with the corresponding lowercase letter.
`descendantClassEnum(<String>)`	Shows the children of the class entered as a string parameter.
`enum(sCompletionString1,sCompletionString2,...)`	List of comma-delimited strings suggested for completion at the template invocation.
`escapeString(sEscapeString)`	Escapes the specified string.
`expectedType()`	Returns the type which is expected as a result of the whole template. Makes sense if the template is expanded in the right part of an assignment, after return, etc.
`fileName(sFileName)`	Returns file name with extension.
`fileNameWithoutExtension()`	Returns file name without extension.
`firstWord(sFirstWord)`	Returns the first word of the string passed as a parameter.
`groovyScript("groovy code")`	Returns Groovy script with the specified code. You can use `groovyScript` macro with multiple arguments. The first argument is a script text that is executed or a path to the file that contains a script. The next arguments are bound to `_1, _2, _3, ..._n` variables that are available inside your script. Also, `_editor` variable is available inside the script. This variable is bound to the current editor.
`guessElementType (<container>)`	Makes a guess on the type of elements stored in a `java.util.Collection`. To make a guess, IntelliJ IDEA tries to find the places where the elements were added to or extracted from the container.
`iterableComponentType(<ArrayOrIterable>)`	Returns the type of an iterable component, such as an array or a collection.
`iterableVariable()`	Returns the name of a variable that can be iterated.
`lineNumber()`	Returns the current line number.
`lowercaseAndDash(String)`	Returns lower case separated by dashes, of the string passed as a parameter. For example, the string `MyExampleName` is converted to `my-example-name`.
`methodName()`	Returns the name of the embracing method (where the template is expanded).
`methodParameters()`	Returns the list of parameters of the embracing method (where the template is expanded).
`methodReturnType()`	Returns the type of the value returned by the current method (the method within which the template is expanded).
`qualifiedClassName()`	Returns the fully qualified name of the current class (the class where the template is expanded). Tip Clear the Shorten FQ names check box.
`rightSideType()`	Declares the left-side variable with a type of the right-side expression. It is used in the iterations group to have a single template for generating both raw-type and Generics Collections.
`snakeCase(sCamelCaseText)`	Returns snake_case string out of CamelCase string passed as a parameter.
`spaceSeparated(String)`	Returns string separated with spaces out of CamelCase string passed as a parameter. For example, if the string passed as a parameter is `fooBar`, then the function returns `foo bar`.
`subtypes(sType)`	Returns the subtypes of the type passed as a parameter.
`suggestIndexName()`	Suggests the name of an index variable. Returns `i` if there is no such variable in scope, otherwise returns `j` if there is no such variable in scope, etc.
`suggestVariableName()`	Suggests the name for a variable based on the variable type and its initializer expression, according to your code style settings that refer to the variable naming rules. For example, if it is a variable that holds an element within iteration, IntelliJ IDEA makes a guess on the most reasonable names, also taking into account the name of the container being iterated.
`suggestFirstVariableName(sFirstVariableName)`	Doesn't suggest `true, false, this, super`.
`time(sSystemTime)`	Returns the current system time.
`typeOfVariable(VAR)`	Returns the type of the variable passed as a parameter.
`underscoresToCamelCase(sCamelCaseText)`	Returns the string passed as a parameter with CamelHump letters substituting for underscores. For example, if the string passed as a parameter is `foo_bar`, then the function returns `fooBar`.
`underscoresToSpaces(sParameterWithSpaces)`	Returns the string passed as a parameter with spaces substituting for underscores.
`user()`	Returns the name of the current user.
`variableOfType(<type>)`	Suggests all variables that may be assigned to the type passed as a parameter, for example variableOfType("java.util.Vector"). If you pass an empty string ("") as a parameter, suggests all variables regardless of their types.
`JsArrayVariable`	Returns JavaScript array name.
`jsClassName()`	Returns the name of the current JavaScript class.
`jsComponentType`	Returns the JavaScript component type.
`jsMethodName()`	Returns the name of the current JavaScript method.
`jsQualifiedClassName`	Returns the complete name of the current JavaScript class.
`jsSuggestIndexName`	Returns a suggested name for an index.
`jsSuggestVariableName`	Returns a suggested name for a variable.

File Types

File | Settings | Editor | File Types for Windows and Linux

IntelliJ IDEA | Preferences | Editor | File Types for OS X

Ctrl+Alt+S

Use this page to manage the list of file types and extension patterns to be recognized by IntelliJ IDEA.

Item	Keyboard Shortcut	Description
Recognized file types		This list box displays all the default and custom file types currently supported by IntelliJ IDEA. Refer to the section File Types Recognized by IntelliJ IDEA . Use the Add, Edit, and Remove buttons to manage the contents of the list box. Tip Default types cannot be edited or removed.
+	Alt+Insert	Click this button to open the New File Type dialog and define a new custom file type there.
	Enter	Click this button to open the Edit File Type dialog box and edit the selected file type there. This button is disabled when a default file type is selected.
−	Alt+Delete	Click this button to delete the selected file type from the list. This button is disabled when a default file type is selected.
Registered Patterns		Shown in this area are all the registered extensions associated with the file type selected in the Recognized file types list. Use +, and − to manage the corresponding patterns.
+	Alt+Insert	Use this icon or shortcut to open the Add wildcard dialog box and specify a new pattern using wildcards there.
	Enter	Use this icon or shortcut to edit the selected pattern.
−	Alt+Delete	Use this icon or shortcut to remove the selected pattern from the list.
Ignore files and folders		In this text box, specify the files and folders, which you want to be ignored by IntelliJ IDEA. Such files and folders will be completely excluded from any kind of processing. By default the list includes temporary files, service files related to version control systems, etc. You can specify multiple names or wildcard masks, with semicolons (;) as separators. Below is the default setting, in case you need to restore it: `CVS;SCCS;RCS;rcs;.DS_Store;.svn;.pyc;.pyo;*.pyc;*.pyo;.git;*.hprof;_svn;.hg;*.lib;*~;__pycache__;.bundle;*.rbc;*$py.class;`

New File Type

File | Settings | Editor | File Types - Add/Edit for Windows and Linux

IntelliJ IDEA | Preferences | Editor | File Types - Add/Edit for OS X

Ctrl+Alt+S

The dialog box opens when you click the Add button or select a custom file type and click the Edit button on the File Types page .

Use the dialog box to configure and re-configure presentation and highlighting of keywords, comments, numbers etc. in files of a specific custom file type. These settings make the basis for parsing files of this type in the editor.

Item	Description
Name	In this text box, specify the name of the file type.
Description	In this text box, provide an optional description of the file type.
Syntax Highlighting	In this area, specify the character strings to indicate borders of comments, the numeric system used, and configure highlighting for brackets, braces, etc. syntax elements.
Line comment	In this text box, specify the character string to indicate the start of a single-line comment.
Only at line start	If this check box is selected, the character string that denotes the beginning of a line comment, is recognized as a comment if located in the first position of a line. This check box becomes available if at least one character is inserted in the Line comment field.
Block comment start	In this text box, specify the character string to indicate the start of a block comment.
Block comment end	In this text box, specify the character string to indicate the end of a block comment.
Hex prefix	In this text box, specify the character string to indicate that the subsequent value is a hexadecimal number. For example, `0x` .
Number postfixes	In this text box, specify the character string to indicate which numeric system or unit is used. A postfix is a trailing string of characters, for example, `e-3, kg` etc.
Support paired braces	Select this check box, to have paired braces highlighted.
Support paired brackets	Select this check box, to have paired brackets highlighted.
Support paired parens	Select this check box, to have paired parentheses highlighted.
Support string escapes	Select this check box, to have string escapes highlighted.
Ignore case	Select this check box to have IntelliJ IDEA ignore case when processing file type extensions.
Keywords	Use this area to flexibly configure highlighting of keywords by grouping them into sets and associating each set with its own highlighting scheme . The area consists of 4 tabs. In each tab, create a set of keywords using the Add and Remove buttons. To associate a keyword set 1-4 with a highlighting scheme, edit the corresponding Keyword1 - Keyword4 property on the Custom page of the Colors and Fonts dialog box.
+	Click this button to open the Add a new keyword dialog box and define the new keyword there.
−	Click this button to delete the selected keyword from the list.

Copyright

File | Settings | Editor | Copyright for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Copyright for OS X

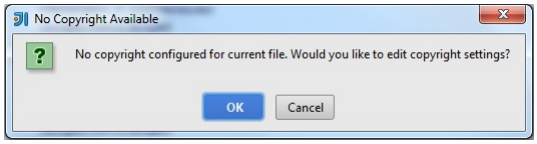
Ctrl+Alt+S

Use this page to associate copyright profiles with the specific scopes within your project.

In this section:

– Copyright
  – Copyright options
– Copyright Profiles
– Formatting
  – File Types

Copyright options

Item	Tooltip and shortcut	Description
Default project copyright		Use this drop-down list to specify the default copyright profile at the project level. The default profile applies to those project files that do not belong to any defined scope. The default profile also applies when no specific profile-scope combinations have been created. By default, IntelliJ IDEA suggests No copyright. If No copyright value is selected, then, on an attempt to create a copyright notice in a file out of any defined scope, IntelliJ IDEA will show a dialog box asking you whether you want to edit the copyright settings.  Choosing OK results in opening this page, where you can select the default project profile from the existing ones, or create a new one; choosing Cancel generates no copyright notice at all. The drop-down list contains the available copyright profiles defined in the Copyright Profiles page. If only No copyright option is available, the scope-profile combinations area is disabled. With at least one copyright profile defined, the scope-profile combinations area below becomes enabled.
Scope-profile combinations area		Note that unless you have at least one copyright profile and scope , the controls below are disabled.
Scope		Click an entry in this column to select the desired shared scope from the drop-down list. If the desired scope is not yet defined, click the link below to open the Scopes dialog box, where you can create a new scope or edit an existing one.
Copyright		Click an entry in this column to select a copyright profile from the drop-down list of available copyright profiles.
+	Add Alt+Insert	Use this icon or shortcut to add a new scope-profile combination.
–	Delete Alt+Delete	Use this icon or shortcut to remove the selected item from the list.
↑↓	Up/Down Alt+Up / Alt+Down	Use these icons or shortcuts to move the selected item one line up or down in the list. The order of scope-profile combination is important, because if a file belongs to several scopes, these scopes are checked upside down. So doing, the first scope where this file belongs, is declared the proper one, and its copyright profile is used.
Select Scopes to add new scopes or modify existing ones		Click this link to open the Scopes page , where you can create a new scope or edit an existing one.

Copyright Profiles

File | Settings | Editor | Copyright | Copyright Profiles for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Copyright | Copyright Profiles for OS X

`Ctrl+Alt+S`

Use this page to create, edit, and remove copyright profiles.

In this topic:

– Profiles Toolbar

– Copyright Profile Page

Profiles Toolbar

Item	Tooltip and Shortcut	Description
	`Insert`	Click this button to add a new profile at the desired level. You can choose to create an empty profile or a profile based on the current default profile.
	`Delete`	Click this button to discard the selected profile.
	`Ctrl+D`	Click this button to open the Copy Copyright Profile dialog box, where you can create a copy of the selected profile.
	Import	Click this button to import a file that contains the desired copyright notice definition.

Tip You can perform the same actions using context menus of the profile nodes.

Copyright Profile Page

Use this page to configure the selected profile: define the copyright notice to be generated and specify the keyword to detect copyright notices in comments.

Item	Description
Name	Use this text field to view or edit the name of the selected copyright profile.
Copyright text (may contain Velocity templates)	Use this text area to view or edit the copyright notice to be generated. A copyright profile can contain an explicit plain text of the copyright notice or its definition through a Velocity template. Currently the following variables are available in the Velocity context:

Name	Type	Comment
`$today`	`DateInfo`	The current date and time.
`$file.fileName`	`String`	The name of the currently opened file where the notice is to be generated.
`$file.pathName`	`String`	The complete path and name of the currently opened file where the notice is to be generated.
`$file.className`	`String`	The name of the currently opened Java file where the notice is to be generated.
`$file.qualifiedClassName`	`String`	The fully qualified name of the currently opened Java file where the notice is to be generated.
`$file.lastModified`	`DataInfo`	The date and time when the current file was last changed.
`$project.name`	`String`	The name of the current project.
`$module.name`	`String`	The name of the current module.
`$username`	`String`	The name of the current user.
`DateInfo` has the following properties:		
`year`	`int`	The current year.
`month`	`int`	The current month (1-12).
`day`	`int`	The current date of month (1-31).
`hour`	`int`	The current hour (0-11).
`hour24`	`int`	The current hour (0-23).
`minute`	`int`	The current minute of the hour (0-59).
`second`	`int`	The current second of the minute (0-59).
`DateInfo` has the following method:		
`format(String format)`	`String`	See java.text.SimpleDateFormat format options.

Validate	Click this button to check that the Velocity template has been specified correctly.
Regex to detect copyright in comments	Use this text box to type, view, or edit the regular expression that will be used to find copyright notices in comments. Tip This regular expression should match the above specified copyright notice. Otherwise instead of updating copyright notices, IntelliJ IDEA will insert new ones.

Allow replacing copyright if old copyright contains:	Use this text box to type or edit the string that will be recognized in the existing copyright notice, and allow replacing it with the new one.

Formatting

File | Settings | Editor | Copyright | Formatting for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Copyright | Formatting for OS X

Ctrl+Alt+S

Use this page to configure common formatting options, regardless of the type of a particular file.

Item	Description
Comment Type	In this area, specify the type of comment to enclose copyright notices in. The available options are: – Use block comment - select this option to have copyright notices enclosed in block comments. – Prefix each line - select this check box to have each line of a copyright notice prepended with the character defined in the Separator char text box. By default, an asterisk is used. – Use line comment - select this option to have copyright notices represented as a sequence of line comments.
Borders	In this area, define how to separate copyright notices from other comments. – Separator before - select this check box have a copyright notice prepended with a line of characters defined in the Separator char text box. The number of characters in a separator line is defined in the Length text box. – Separator after - select this check box have a copyright notice followed by a line of characters defined in the Separator char text box. The number of characters in a separator line is defined in the Length text box. – Separator char - in this text box, type the character that will be used in the separator strings and as a prefix or ending character in block comments. – Box - select this check box to have each line of a copyright notice followed by a character defined in the Separator char text box. – Add blank line after - select this check box to have a blank line inserted after a copyright notice.
Relative Location	In this area, specify the location of a copyright notice relative to other comments in a file. – Before other comments - when this option is selected, copyright notices are inserted above other comments. – After other comments - when this option is selected, copyright notices are inserted below other comments.
Preview	Use this area to view a sample copyright notice with the formatting you have defined.

File Types

File | Settings | Editor | Copyright | Formatting | <File Type> for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Copyright | Formatting | <File Type> for OS X

Ctrl+Alt+S

The page opens when you click a file type below the Formatting node. Use this page to define formatting of copyright notices depending on the target file type.

Item	Description
No copyright	If this option is selected, the copyright notice will not be updated in the files of the selected type.
Use default settings	If this option is selected, the copyright notice will be updated according to the default project settings.
Use custom formatting options	If this option is selected, the copyright notice in the files of the selected type will be updated according to the custom settings defined below.
Comment Type	In this area, specify the type of comment to enclose copyright notices in. The available options are: – Use block comment - select this option to have copyright notices enclosed in block comments. – Prefix each line - select this check box to have each line of a copyright notice prepended with the character defined in the Separator char text box. By default, an asterisk is used. – Use line comment - select this option to have copyright notices represented as a sequence of line comments.
Borders	In this area, define how to separate copyright notices from other comments. – Separator before - select this check box have a copyright notice prepended with a line of characters defined in the Separator char text box. – Separator after - select this check box have a copyright notice followed by a line of characters defined in the Separator char text box. – Separator char - in this text box, type the character that will be used in the separator strings and as a prefix or ending character in block comments. – Length - in this text box, type the number of characters in a spartor line. – Box - select this check box to have each line of a copyright notice followed by a character defined in the Separator char text box. – Add blank line after - select this check box to have a blank line inserted after a copyright notice.
Relative Location	In this area, specify the location of a copyright notice relative to other comments. – Before other comments - when this option is selected, copyright notices are inserted above other comments. – After other comments - when this option is selected, copyright notices are inserted below other comments.
Preview	Use this area to view a sample copyright notice with the formatting you have defined.

Location in File	Use this area to specify the location of the copyright notice in a file. Depending on the file type, the available options are:
	– JAVA
	  – Before the package statements
	  – Before imports
	  – Before the class declaration
	– HTML, JSP, JSPX, XML
	  – Before the root tag
	  – Before the Doctype statement
	– For Properties, PHP, JavaScript, CSS, SCSS, and SASS file types no choice is available. Copyright notices are always inserted at the top of a file.

Emmet

File | Settings | Editor | Emmet for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Emmet for OS X

Ctrl+Alt+S

On the nested pages, enable IntelliJ IDEA to use Emmet in HTML, XML , JavaScript (JSX Harmony) files and style sheets .

ItemDescription	
Expand abbreviation with	Use this drop-down box to select the default key to expand Emmet selectors with. This key will also by default expand Emmet live templates .

Note IntelliJ IDEA expands abbreviations only if their output does not exceed 15 KB.

CSS

File | Settings | Editor | Emmet - CSS for Windows and Linux

for OS X

`Ctrl+Alt+S`

Item	Description
Enable CSS Emmet	Select this check box to enable Emmet support for style sheets. If this check box is not selected, the complicated abbreviations, like `bdci:n` expanding into `border-corner-image: none;` , will not work in the editor.
Enable fuzzy search among CSS abbreviations	When this check box is selected, every unknown abbreviation will be scored against available template names. The match with the best score will be used to resolve the template. For example, with this option enabled, the following abbreviations can be equal to: – `ov:h` – `ov-h` – `o-h` – `oh`
Enable expansion of unknown properties ('unknown' to 'unknown:;')	– When this check box is selected, any entered word will be expanded into the same word followed with a colon and a semicolon; – When this check box is cleared, only known properties (for example, `color`) will be expanded this way (`color:;`)
Auto insert CSS vendor prefixes	If this check box is selected, the CSS properties listed in the table below are expanded into constructs that contain pre-pending vendor prefixes. Learn more at Vendor prefixes . If this check box is cleared, the entire table of properties is disabled.
Properties and vendor prefixes	The table contains a list of CSS properties and vendor prefixes that correspond to various browsers. – To enable or disable a property in a browser, select or clear the check box under the browser column. – To add a new property to the list, click the Add button + or press `Alt+Insert` . Then type the name of the property in the dialog box that opens and enable it in the relevant browsers. – To delete one or more properties from the lsit, select them and press Remove – or press `Alt+Delete` .

JSX

File | Settings | Editor | Emmet - JSX for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Emmet - JSX for OS X

Ctrl+Alt+S

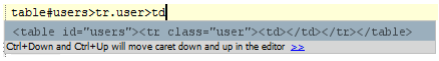
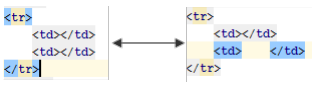
Item	Description
Enable JSX Emmet	Select this check box to use Emmet within XML fragments in the JSX Harmony context if you have chosen the JSX language level on the JavaScript page of the Settings dialog box.

HTML

File | Settings | Editor | Emmet - HTML for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Emmet - HTML for OS X

Ctrl+Alt+S

ItemDescription	
Enable XML/HTML Emmet	Select this check box to enable Emmet support for XML and HTML. If this check box is not selected, complicated abbreviations, such as `div.class>ul#list>.item$)` and similar, will not work in the editor.
Enable abbreviation preview	Select this check box to have IntelliJ IDEA show a pop-up window with a preview of the entered abbreviation before actually expanding it . 
Enable automatic URL recognition while wrapping text with <a> tag	– If this check box is cleared and you attempt to wrap an URL address with the <a> tag, IntelliJ IDEA simply encloses the URL address in `<a href=""></a>` and positions the cursor inside the double quotes in the `href` attribute. For example, wrapping `http://www.jetbrains.com` will result in `<a href="` | `">http://www.jetbrains.com</a>` : – If this check box is selected and you attempt to wrap an URL address with the <a> tag, IntelliJ IDEA inserts the URL address inside the double quotes as the value of the `href` attribute and encloses the URL in `<a href="<wrapped URL>"></a>` . For example, wrapping `http://www.jetbrains.com` will result in `<a href="http://www.jetbrains.com">http://www.jetbrains.com</a>` . Moreover, IntelliJ IDEA highlights the wrapped URL green as a recognized URL:
Add edit point at the end of template	If this check box is selected, an editing position adds to the end of an HTML template (END); if this check box is not selected, then the new edit point is not added. Compare the following: 
BEM	In this area, specify the BEM separators for the class names, modifiers and short elements. Refer to the Emmet documentation for details.
Filters enabled by default	In this area, specify which Emmet filters you want to be applied to an expanded abbreviation before it is shown in the editor. Learn more about filters at http://docs.emmet.io/filters/ . To have a filter always applied by default, select the check box next to it. The available options are: – XSL tuning – Comment tags – Escape – Single line – BEM – Trim line markers

GUI Designer

File | Settings | Editor | GUI Designer for Windows and Linux

IntelliJ IDEA | Preferences | Editor | GUI Designer for OS X

`Ctrl+Alt+S`

Item	Description
Generate GUI into	This option specifies what kind of output the GUI Designer generates for the visual forms you create. The available settings are: – Binary class files. This is the default option. When selected, no Java source code is generated for GUI forms and components. When the project compiles, IntelliJ IDEA simply creates the necessary compiled runtime classes. – Java source files.If this option is selected, the GUI Designer writes Java source code for the form and its components to the source file of the class to which the form is bound , on compiling, running or debugging. During compilation, two blocks of code are added to the form's class: – A private method `$$$setupUI$$$()` that contains the GUI initializer code for the bound form and its components. – A call to the `$$$setupUI$$$()` method. **Warning!** Do not change the generated method, or call it from any other code. If you manually modify GUI initializer code, your UI will no longer be in sync with the `.form` file, and the next compilation will overwrite your changes.
Automatically copy form runtime classes to the output directory	If this option is checked, the classes from the package `com.intellij. uiDesigner.core` package are copied to the configured output directory, when the project is compiled. These classes are used when working with the GridLayoutManager(IntelliJ), or when there are components with mnemonics , specified by the `&` character.
Default Layout Manager	Set the default layout manager for new components placed into forms. The selection here appears as the setting for the *Layout Manager* property whenever a new component is placed on a form. – BorderLayout: Design-time behavior in forms emulates Java's *Border* layout manager. – CardLayout: Design-time behavior in forms emulates Java's *Card* layout manager. – FlowLayout: Design-time behavior in forms emulates Java's *Flow* layout manager. – FormLayout (JGoodies): Design-time behavior in forms emulates JGoodies Forms layout manager. (For more information, see https://jgoodies.dev.java.net/) – GridBagLayout: Design-time behavior in forms emulates Java's *Grid Bag* layout manager. – GridLayoutManager (IntelliJ): Design-time behavior in forms is controlled by this custom layout manager. It's basically a simple grid layout scheme that's sufficient for many uses. It is the default layout manager in new IntelliJ IDEA installations.
Default accessibility for UI-bound fields	Use this option, if you want to change the default accessibility for UI-bound fields from `private` to something else, like `public` .
Resize column and row headers with mouse	This check box enables/disables resizing in captions. If this check box is selected, IntelliJ IDEA allows to resize column and rows using mouse. When pointing to a column or row, the mouse pointer changes its shape to the double arrow.

Images

File | Settings | Editor | Images for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Images for OS X

Ctrl+Alt+S

Specify how you want images to be shown in IntelliJ IDEA. Optionally, specify an external editor for working with images.

Item	Description
Editor	Use this area to specify the settings according to which images should be displayed in IntelliJ IDEA.
Show Grid lines by default	Select this check box to have a grid displayed when reviewing image files. Tip When the check box is selected, the area below is enabled where you can define how to display a grid and its elements.
Show Grid lines only when zoom factor equal or more than	Use this spin box to specify the minimum zoom factor to have a grid displayed.
Show Grid line after every (pixels)	Use this spin box to specify the number of pixels between a pair of grid lines.
Grid line color	From this palette, click the colour to display grid lines.
Show transparency chessboard by default	Select this check box to have transparent pieces of images shown as a chessboard.
Chessboard cell size	Use this spin box to specify the chessboard cell size.
Color of 'white' cell	From this palette, click the colour to display chessboard cells located at initially 'white' positions.
Color of 'black' cell	From this palette, click the colour to display chessboard cells located at initially 'black' positions.
Zoom image with mouse wheel (Ctrl+Mouse Wheel)	Select this check box to enable zooming the image through the Ctrl+Mouse wheel combination.
Enable smart zooming for small images	Select this check box to have small images opened with the zooming factor to the size specified below.
Preferred minimum width/height for smart zooming (pixels)	Use this spin box to set the minimum number of pixels to zoom the small images to.
External Editor	In this area, specify an external editor for working with images.
Executable path	In this text box, specify the path to the executable image editor file.

Intentions

File | Settings | Editor | Intentions for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Intentions for OS X

`Ctrl+Alt+S`

Use this page to enable and disable intention actions .

In this topic:

– Intention List
– Toolbar and Controls

Intention List

The list shows all intention actions currently available in IntelliJ IDEA.

The intention actions are grouped by languages. To enable an intention action, select the check box to its left.

Toolbar and Controls

Item	Tooltip and shortcut	Description
	`Ctrl+NumPad Plus`	Expand all nodes in the intention list.
	`Ctrl+NumPad –`	Collapse all nodes in the intention list.
		Use this text box to search through the list of intention actions. As you type a search string, the intention actions that match the search pattern are displayed. To finalize the search, press `Enter`. The previously used search patterns are stored in the search history list.
		Click this button to clear the search history list.
Description		This read-only field shows the description of the selected intention action.
Usage examples		This area illustrates the effect of applying the selected intention action through the following fields: – Before - this read-only field shows an example of source code before applying the selected intention action. – After - this read-only field shows the result of applying the selected intention action to the above example of source code.

Language Injections

File | Settings | Editor | Language Injections for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Language Injections for OS X

`Ctrl+Alt+S`

For the Language Injections page to be available, the IntelliLang plugin must be enabled.

Use this page to manage the list of available language injections and to configure the language injection feature for text, attributes, and parameters.

To sort the information, use the cells in the header row (Name, Language and Scope). To perform other tasks, use the check boxes and the toolbar icons.

In this section:

- Language Injections
  - Prerequisites
  - Injection entries
  - Toolbar
- Language Injection Settings Dialogs
- Advanced

Prerequisites

Before you start working with language injections, make sure that the IntelliLang plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Injection entries

You can sort the information by any of the columns by clicking the cells in the header row. The current sorting status is shown by the corresponding sorting marker: ▲ for the ascending order or ▼ for the descending order.

Item	Description
Leftmost column	Use the check boxes to enable or disable the corresponding injections. You can also enable or disable a number of injections at once. To do that, select the required injections in the list and click Enable Selected Injections or Disable Selected Injections on the toolbar.
Name	This column shows the names of the injections and the libraries that implement them.
Language	This column shows the languages to which the corresponding injections pertain.
Scope	One of the following: – Built-in. This is a category for pre-defined injections. In terms of the scope, those are IDE-level injections. – IDE. User-defined injections that are available in all of your projects. – Project. User-defined injections that are available only in the current project. You can move the user-defined injections between the IDE and the project levels by using on the toolbar.

Toolbar

Item	Tooltip and Shortcut	Description
+	Add `Insert`	Use this icon or shortcut to create a new entry. Select the target category from the list and then specify the settings in the dialog that opens.
–	Remove `Delete`	Use this icon or shortcut to remove the selected entries from the list.
	Edit `Enter`	Click this icon to re-configure the selected entry in the Language Injection Settings dialog box that opens.
	Duplicate	Use this icon to create a copy of the selected injection entry. Edit the copy as necessary.
	Enable Selected Injections	Click this icon to enable the injections that are currently selected in the list.
	Disable Selected Injections	Click this icon to disable the injections that are currently selected in the list.
	Move to Project/IDE Scope `Shift+Space`	Use this icon or shortcut to move the selected injections between the IDE and the project levels. See also, Scope.
	Import	Click this button to have an injection configuration from another IntelliJ IDEA installation imported. The Select Path dialog opens, where you can select the `IntelliLang.xml` file from the desired location. The dialog box that opens displays the entries contained in the configuration file. Remove the unnecessary entries from the list using the Delete button. This will not affect the contents of the configuration file.

This selective import-feature makes it easy to share certain configurations in a team without losing any local entries as it happens when settings are imported via the core Importing Settings feature.

To prevent inconsistent data, import is only possible if the existing configuration has not been changed or has been saved using the Apply button.

Export

Click this button to have one or several selected injection configurations saved in a file. Upon clicking this button, the Export Selected Injections to File dialog box opens.

- To have the configurations appended to an existing file, select the required file.
- To have the configurations saved in a new file, specify the file name and choose the file type from the drop-down list.

Language Injection Settings dialogs

- Language Injection Settings Dialog: Generic Groovy
- Language Injection Settings Dialog: Generic JavaScript
- Language Injection Settings Dialog: Java Parameter
- Language Injection Settings Dialog: XML Attribute Injection
- Language Injection Settings Dialog: XML Tag Injection

Language Injection Settings dialog: Generic Groovy

File | Settings | Language Injection - Groovy

Ctrl+Alt+S

The dialog opens when you click + on the Language Injections page, and choose Generic Groovy on the context menu, or select an entry and click .

IntelliJ IDEA comes with a set of predefined injection configurations which is quite sufficient to ensure high productivity and comfortable environment. Therefore it is strongly recommended that you use the predefined injection configurations and avoid creating new ones.

Item	Description
Name	In this text box, type the name that will identify the injection configuration in the list. The text box is available when configuration is created (+), or configured ( ).
Language	In this area, specify the language of the injection. If necessary, define the context to be automatically made up when the injection is opened in the editor . – ID - from this drop-down list, select the ID of the language of the injection. – Prefix - in this text box, specify the character string to be automatically added before the injection. – Suffix - in this text box, specify the character string to be automatically added after the injection. The Prefix and Suffix text boxes are optional.
Places Pattern	In this text box, type the rules that define the context where you want IntelliJ IDEA recognize literals as injections. Note The rules are built from Program Structure Interface Patterns and are actually chained calls of methods of an internal IntelliJ IDEA language. The Program Structure Interface shows the structure of a file as IntelliJ IDEA treats it. For more information on the syntax used, refer to Custom Plugin Development guide. Warning! These rules are IntelliJ IDEA internals, and it is strongly recommended that you use the predefined injection configurations and avoid creating new ones.
Advanced	In this area, specify additional settings to narrow the context where the injection is applicable and thus the enable more fine-grained control over the injection process. – Value pattern - in this text box, type a regular expression that determines the context to inject the language into. By using the first capturing group of the pattern as the target for injection, you can configure the procedure to have the language injected only into values that match a certain pattern or into multiple parts that match the pattern. For example, `^javascript:(.*)` matches the `javascript` protocol that can be used in hyperlink-hrefs to execute JavaScript code. – Single file - select this check box to specify an exact file to apply the injection to. Type the name of the file in the Value pattern.

Language Injection Settings dialog: Generic JavaScript

File | Settings | Editor | Language Injection - Generic JavaScript for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Language Injection - Generic JavaScript for OS X

Ctrl+Alt+S

The dialog box opens when you click + and choose Generic JS on the context menu, or select an entry and click .

Item	Description
Name	In this text box, type the name that will identify the injection configuration in the list. The text box is available when configuration is created (+), or configured ( ).
Language	In this area, specify the language of the injection. If necessary, define the context to be automatically made up when the injection is opened in the editor . – ID - from this drop-down list, select the ID of the language of the injection. – Prefix - in this text box, specify the character string to be automatically added before the injection. – Suffix - in this text box, specify the character string to be automatically added after the injection. The Prefix and Suffix text boxes are optional.
Places Pattern	In this text box, type the rules that define the context where you want IntelliJ IDEA recognize literals as injections. Note The rules are built from Program Structure Interface Patterns and are actually chained calls of methods of an internal IntelliJ IDEA language. The Program Structure Interface shows the structure of a file as IntelliJ IDEA treats it. For more information on the syntax used, refer to Custom Plugin Development guide. Warning! These rules are IntelliJ IDEA internals, and it is strongly recommended that you use the predefined injection configurations and avoid creating new ones.
Advanced	In this area, specify additional settings to narrow the context where the injection is applicable and thus the enable more fine-grained control over the injection process. – Value pattern - in this text box, type a regular expression that determines the context to inject the language into. By using the first capturing group of the pattern as the target for injection, you can configure the procedure to have the language injected only into values that match a certain pattern or into multiple parts that match the pattern. For example, `^javascript:(.*)` matches the `javascript` protocol that can be used in hyperlink-hrefs to execute JavaScript code. – Single file - select this check box to specify an exact file to apply the injection to. Type the name of the file in the Value pattern.

Language Injection Settings dialog: Java Parameter
File | Settings | Language Injection - Java Parameter

Ctrl+Alt+S

The dialog box opens when you click + on the Language Injections page, and choose Java Parameter on the context menu, or select an entry and click .

Use this dialog box to configure language injections for Java parameters. The dialog box provides the ability to make use of IntelliLang's features, if, for any reason, the injection annotations cannot be used. This mainly applies to configuring third party/library methods as well as projects that still have to use Java 1.4.

Item	Description
Language	In this area, specify the language to inject and the injection context. – ID - from this drop-down list, select the ID of the language to inject. – Prefix - in this text box, specify a prefix to make up the injection context. – Suffix - in this text box, specify a suffix to make up the injection context. Note that the Prefix and Suffix fields are optional.
Class Methods	Click … and select the class of interest. As a result, the methods of the class along with their parameters are shown. Select the parameters for which you want to enable the injection.
Advanced	Specify additional settings. – Value pattern - type a regular expression for the injection context. – Single file - select this check box to specify an exact file to apply the injection to.

Language Injection Settings dialog: XML Attribute Injection

File | Settings | Language Injection - XML Attribute for Windows and Linux

IntelliJ IDEA | Preferences | Language Injection - XML Attribute for OS X

Ctrl+Alt+S

The dialog box opens when you click + on the Language Injections page, and choose XML Attribute Injection on the context menu, or select an entry and click .

Item	Description
Name	In this text box, type the name that will identify the injection configuration in the list. The text box is available when configuration is created (+), or configured ( ).
Language	In this area, specify the language of the injection. If necessary, define the context to be automatically made up when the injection is opened in the editor . – ID - from this drop-down list, select the ID of the language of the injection. – Prefix - in this text box, specify the character string to be automatically added before the injection. – Suffix - in this text box, specify the character string to be automatically added after the injection. The Prefix and Suffix text boxes are optional.
XML Tag	In this area, specify the XML tags in which the attributes are impacted by the defined configuration. – Local name - in this text box, specify the tag name without a namespace prefix. Use regular expressions to specify multiple tag names (`name1|name2`), case-insensitive names (`(?i)tagname` matches *tagname* as well as *TagName*), etc. Warning! Space characters are not allowed as they affect the match result. – Namespace - in this text box, specify the namespace URI of the XML tag. Both fields are optional. However, if the Local name text box is empty the configuration will apply to any attribute that matches the configured name, regardless of its containing XML tag.
XML Attribute	In this area, define the XML tag attribute which indicates that the text enclosed in a tag with such attribute should be treated as the selected language. – Local name - in this text box, specify the attribute name without a namespace prefix. Use regular expressions: For example, to match HTML event handler attributes, type `on.*` in the text box. The field is optional, unless the Local name text box in the XML Tag area is empty. If the attribute local name is not specified, the configuration applies to all attributes of the enclosing tag. – Namespace - in this text box, specify the namespace URI of the attribute.
Advanced	In this area, specify additional settings to enable more fine-grained control over the injection process. – Value pattern - in this text box, type a regular expression that determines the part of the attribute's value to inject the language into. By using the first capturing group of the pattern as the target for injection, you can configure the procedure to have the language injected only into values that match a certain pattern or into multiple parts that match the pattern. – XPath condition - in this text box, specify an XPath expression to address the injection-target more precisely. The context in which the expression is evaluated is the attribute itself. For the field to be active, the XPathView + XSLT Support plugin must be enabled. – Single file - select this check box to specify an exact file to apply the injection to.

Language Injection Settings dialog: XML Tag Injection

File | Settings | Language Injection - XML Tag for Windows and Linux

IntelliJ IDEA | Preferences | Language Injection - XML Tag for OS X

`Ctrl+Alt+S`

The dialog box opens when you click + on the Language Injections page, and choose XML Tag on the context menu, or select an entry and click .

Item	Description
Name	In this text box, type the name that will identify the injection configuration in the list. The text box is available when configuration is created (+), or configured ( ).
Language	In this area, specify the language of the injection. If necessary, define the context to be automatically made up when the injection is opened in the editor . – ID - from this drop-down list, select the ID of the language of the injection. – Prefix - in this text box, specify the character string to be automatically added before the injection. – Suffix - in this text box, specify the character string to be automatically added after the injection. The Prefix and Suffix text boxes are optional.
XML Tag	In this area, define the XML tag which indicates that the text enclosed in this tag should be treated as the selected language. – Local name - in this text box, specify the tag name without a namespace prefix. Use regular expressions to specify multiple tag names (`name1|name2`), case-insensitive names (`(?i)tagname` matches *tagname* as well as *TagName*), etc. **Warning!** Be sure not to enter any space characters as they would be significant for the match. – Namespace - in this text box, specify the namespace URI of the XML tag. The field is optional.
Sub-tags	Select this check box to include all the subtags recursively.
Advanced	In this area, specify additional settings to enable more fine-grained control over the injection process. – Value pattern - in this text box, type a regular expression that determines the part of the XML text's value to inject the language into. By using the first capturing group of the pattern as the target for injection, you can configure the procedure to have the language injected only into values that match a certain pattern or into multiple parts that match the pattern. Examples: `[$#]\{(.*?)\}` matches the pattern used by the JSP/JSF Expression Language. `^javascript:(.*)` matches the `javascript` protocol that can be used in hyperlink-hrefs to execute JavaScript code. – XPath condition - in this text box, specify an XPath expression to address the injection-target more precisely. The context in which the expression is evaluated is the surrounding XML tag. It is possible to use the XPath extension functions that are provided by the Jaxen XPath engine, e.g. `lower-case()` . Also, there are three additional functions that can be used to determine the current file's name, extension, and file type: `file-name()` , `file-ext()` and `file-type()` . Alternatively, a list of available functions can be retrieved through standard code completion. For the field to be active, the XPathView + XSLT Support plugin must be enabled. **Warning!** For performance reasons, it is recommended that you keep these expressions as simple as possible. Especially expressions that cause the whole document to be scanned, such as `//foo/bar` might cause performance problems with large files. – Single file - select this check box to specify an exact file to apply the injection to.

Advanced

File | Settings | Editor | Language Injections | Advanced for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Language Injections | Advanced for OS X

`Ctrl+Alt+S`

Use this page to specify different names for the base-annotations to be used. This helps avoid any dependencies on foreign code where it is not desired or possible. The custom annotations should provide the same properties as the original ones, i.e. `value` for all of them and an optional (default = "") `prefix` and `suffix` for the `@Language` replacement.

Also configure the runtime checks to be generated for the `@Pattern` validation.

Item	Description
Annotation Classes	In this area, specify the classes that implement annotations of the following types: – Language annotations – Pattern annotations – Substitution annotations Type the class names, possibly using code completion. If necessary, use the Browse button … to open the Select Class dialog box, where you can you can locate the desired class in the project tree view. Alternatively, switch to the Search by Name tab and start typing the class name. As you type, the list of available classes narrows down to match your entry.
Runtime Pattern Validation	In this area, configure the runtime checks to be generated for the `@Pattern` validation. The available options are: – No runtime instrumentation - if this option is selected, no checks will be inserted and any compiled class files will be affected. – Instrument with assertions - if this option is selected, pattern-validation is controlled with the `-ea` JVM switch and throws `AssertionError`. Tip Selecting this option is recommended due to the potentially negative impact on the performance for methods that are invoked very often. – Instrument with IllegalArgumentException - select this option to have the same result as when the `@NotNull` instrumentation of IntelliJ IDEA is used.
Performance	Click one of the radio buttons in this area to select the level of analysis and performance of the language injections resolution procedure. – Do not analyze (fast) - if this option is selected, IntelliJ IDEA does not analyze injections. Tip Selecting the Do not analyze option significantly improves performance. – Analyze references - if this option is selected, IntelliJ IDEA attempts to recognize injections introduced through variables. – Look for variable assignments - if this option is selected, IntelliJ IDEA does not perform the dataflow analysis to detect the substitution strings, but looks for variable assignments only. – Use dataflow analysis (slow) - if this option box is selected, IntelliJ IDEA applies dataflow analysis to language injections.
Convert undefined operands to text in concatenation	If this check box is selected, IntelliJ IDEA inserts an injected operand as a literal, if its type is not recognized.
Add @Language annotation or comment if needed	This check box enables/disables adding annotations or comments.

Spelling

File | Settings | Editor | Spelling for Windows and Linux

IntelliJ IDEA | Preferences | Editor | Spelling for OS X

`Ctrl+Alt+S`

Use this settings page to create your own spelling dictionaries and thus expand the basic spelling support provided by IntelliJ IDEA by default.

On this page:

– Accepted Words Tab

– Dictionaries Tab

Accepted Words Tab

Use this tab to configure the list of words that should be skipped by the Typo inspection.

Item	Tooltip and Shortcut	Description
+	`Alt+Insert`	Click this icon to open the Add New Word dialog box and specify a new entry there. CamelCase or snake_case are not supported. If you try to add a word that is already included in one of the spelling dictionaries, IntelliJ IDEA displays an error message The word <just typed word> is already in the dictionary.
–	`Alt+Delete`	Click this button to delete the selected item from the list.

Dictionaries Tab

Use this tab to configure the dictionaries to be used for spellchecking.

Item	Description
Dictionaries	This area in the bottom of the page shows a list of the dictionaries that can be used in spellchecking. The list contains the dictionaries that come bundled with IntelliJ IDEA by default and user-defined dictionaries detected in the folders from the Custom Dictionaries Folder area above. – To have a default dictionary applied in the current project, select the check box next to it. – To exclude a default dictionary from spellchecking within the scope of the current project, clear the check box next to it.
Custom Dictionaries Folder	This area displays a list of directories that contain user-defined dictionary files (text files with the `dic` extension, containing words separated with a newline). – To add a new folder to the list, click + and choose the required folder in the Select Path Dialog dialog that opens . The full path to the folder is added to the Custom Dictionaries Folder list, and all the `*.dic` files found in this folder are added to the Dictionaries list. – To remove a folder from the list, select it and click –.

TextMate Bundles

File | Settings | Editor | TextMate Bundles for Windows and Linux

IntelliJ IDEA | Preferences | Editor | TextMate Bundles for OS X

Ctrl+Alt+S

Use this page to import the TextMate/SublimeText 2 bundles , and to map color scheme of IntelliJ IDEA to that of TextMate.

This page appears in the Settings/Preferences dialog, when TextMate bundle support plugin is installed and enabled .

The plugin is not bundled with IntelliJ IDEA.

Item	Description
TestMate Bundles	This table of added bundles consists of the following columns: – Check box: If a check box to the left of the added bundle name is selected, the bundle provides highlighting in the files of the corresponding type. – Name of the TextMate Bundle – Bundle location: for each added TextMate bundle, its location is shown.
+	Click this button to locate the desired bundle using the Select Path dialog box. When added, the bundle appears in the table of TextMate Bundles.
–	Click this button to remove the selected bundle. The bundle in question is removed from the list, but is not physically deleted from the computer.
IDE Color Scheme - TextMate Color Scheme	Use this table to establish mapping between the various color schemes of IntelliJ IDEA and TextMate. Tip If you want to use a custom TextMate color scheme, you can import a TextMate bundle with schemes, and it will become visible in the list of TextMate color schemes after clicking Apply.

TODO

File | Settings | Editor | TODO for Windows and Linux

IntelliJ IDEA | Preferences | Editor | TODO for OS X

Ctrl+Alt+S

In this page, configure filters to maintain your lists of TODO items in the TODO tool window and define TODO patterns to be used in filters.

Patterns

In this area, create and manage the list of available TODO patterns.

Item	Description
Icon	This read-only field displays the icon that is assigned to the current pattern. (This icon appears in the TODO tool window as a marker for the corresponding TODO items.)
Case sensitive	This read-only check box indicates whether the current pattern is case sensitive or not. The status is changed in the Edit Pattern dialog box.
Pattern	This read-only field displays the regular expression that describes the TODO pattern. IntelliJ IDEA recognizes regular expressions in the source code against the specified patterns and displays them in the TODO tool window.
+ (Alt+Insert)	Use this icon or shortcut to open the Add Pattern dialog box, where you can create a new ToDo pattern by specifying a regular expression.
(Enter)	Use this icon or shortcut to open the Edit Pattern dialog box, where you can modify the selected pattern.
– (Alt+Delete)	Use this icon or shortcut to remove the selected pattern from the list.

Filters

In this area, manage the list of available filters.

Item	Description
Name	This read-only field shows a list of filter names.
Patterns	This read-only field shows the names of patterns included in the current filter. A filter can contain several patterns.
+ (Alt+Insert)	Use this icon or shortcut to open the Add Filter dialog box, where you can define a new filter.
(Enter)	Use this icon or shortcut to open the Edit Filter dialog box, where you can edit the settings for the selected filter.
– (Alt+Delete)	Use this icon or shortcut to delete the selected filter.

Add / Edit Filter Dialog

File | Settings | TODO - Add/Edit Filter for Windows and Linux

IntelliJ IDEA | Preferences | TODO - Add/Edit Filter for OS X

Ctrl+Alt+S

Use this dialog box to define filters that help track TODO items in your source code.

Item	Description
Name	In this text box, specify the name of the filter.
Patterns	From the list of available patterns, choose the ones to be included in the filter by selecting the check boxes next to them.

Add / Edit Pattern Dialog

File | Settings | TODO - Add/Edit Pattern for Windows and Linux

IntelliJ IDEA | Preferences | TODO - Add/Edit Pattern for OS X

Ctrl+Alt+S

Use this dialog box to define patterns that help track TODO items in your source code. Make sure the TODO items in the source code are inserted inside comments that are valid for the supported file types.

Item	Description
Pattern	In this text box, type the regular expression that describes the desired TODO pattern.
Icon	From this drop-down list, choose an icon for the pattern.
Case sensitive	Select this check-box to make the pattern case-sensitive.
Font type	Select the corresponding check box to have the icon text displayed in bold or italic.
Foreground	Select this check box to enable the palette and choose the foreground color.
Background	Select this check box to enable the palette and choose the background color.
Error Stripe Mark	Select this check box to enable the palette and choose the error stripe color.
Effects	Select this check box to enable effects (underscore, strikethrough, etc.) and choose the color for them from the palette.

Plugins

File | Settings | Plugins for Windows and Linux

IntelliJ IDEA | Preferences | Plugins for OS X

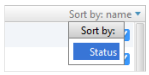
Ctrl+Alt+S

The Plugins page shows the list of installed plugins . Use the check boxes next to plugin names to enable or disable them.

Other controls on this page let you sort and filter the plugin list, update and uninstall repository plugins, access plugin repositories , and also install the plugins available locally.

– Main controls
– Context menu commands
– Colors for plugin statuses

Main controls

Item	Description
	Type the text to be found. As you type, the list of plugins changes: only the plugins whose names and descriptions contain the specified text are shown. To bring the plugin list back to its initial state, delete the text in the search box or click . To access the list of memorized search strings, click .
Show	Use this list to specify which plugins should be shown: – All plugins – Enabled – Disabled – Bundled – Custom (these are the plugins that are not bundled with the IDE).
	If you want to add sorting by plugin status, click and select Status. As a result, disabled plugins will be displayed at the end of the list. This option is also available from the context menu.
Install JetBrains plugin	Click this button to open the Browse JetBrains Plugins Dialog to download and install plugins from the JetBrains repository.
Browse repositories	Click this button to open the Browse Repositories dialog to work with plugin repositories (to download and install repository plugins, manage enterprise plugin repositories, etc.).
Install plugin from disk	Click this button to install a plugin available locally. Select the file that implements the required plugin in the dialog that opens .

Context menu commands

Command	Description
Reload List of Plugins	Use this command to check if newer versions of installed plugins are available (see Colors for plugin statuses).
Sort by | Status	Use this command to sort plugins by their status. When this option is turned on, disabled plugins are shown at the end of the list, after the enabled plugins.
Update Plugin	For repository plugins: use this command to download and install a newer version of the selected plugin (if available). See Colors for plugin statuses .
Uninstall	For plugins that are not bundled with the IDE: use this command to uninstall the selected plugin. Alternatively, the Uninstall button in the plugin description area can be used.

Colors for plugin statuses

The names of plugins are shown in different colors depending on their status.

Color	Plugin status
Black	"Normal" plugin status. For a repository plugin: the plugin version is up-to-date.
Red	One of the following: – The plugin is incompatible with the installed version of IntelliJ IDEA. – The plugin depends on another plugin which is disabled.
Blue	For a repository plugin: a newer version of the plugin is available.
Green	For a repository plugin: the plugin has been downloaded and installed, but has not been activated yet (IntelliJ IDEA needs to be restarted).
Gray	The plugin has been uninstalled, but the changes have not taken effect yet (IntelliJ IDEA needs to be restarted).

Browse JetBrains Plugins dialog

This dialog opens when you click the Install JetBrains plugin button on the Plugins page .

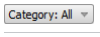
This dialog shows a list of the JetBrains repository plugins .

You can sort and filter the list, download and install plugins, and configure HTTP proxy settings.

– Main controls

– Context menu commands

Main controls

Item	Description
	Type the search string. As you type, the list of plugins changes: only the plugins whose names and descriptions contain the specified string are displayed. If you want IntelliJ IDEA to remember the search string, press Enter . To bring the plugin list back to its initial state, delete the text in the search box or click . To access the list of memorized search strings, click .
	Click this icon to update the list of plugins and their statuses.
	This drop-down list provides category-based filtering. You can select to see all plugins, or only the plugins that belong to a the selected category.
Sort by	Click and select a category.
Install plugin	Click the Install plugin button in the description area to download and install the selected plugin.
HTTP Proxy Settings	If you access the Internet via an HTTP proxy, click this button and specify the HTTP proxy settings .

Context menu commands

Command	Description
Reload List of Plugins	Use this command to update the list of plugins and their statuses.
Sort by | <Category>	Use this command to enable or cancel sorting. The following categories are available: – Status – Downloads – Rating – Last Updated
Download and Install	Use this command to download and install the selected plugin.

Browse Repositories Dialog

This dialog opens when you click the Browse repositories button on the Plugins page .

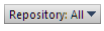
This dialog shows a list of repository plugins .

You can sort and filter the plugins list, download and install plugins, manage enterprise plugin repositories , and configure HTTP proxy settings.

– Main controls

– Context menu commands

Main controls

Item	Description
	Type the search string. As you type, the list of plugins changes: only the plugins whose names and descriptions contain the specified string are displayed. If you want IntelliJ IDEA to remember the search string, press Enter . To bring the plugin list back to its initial state, delete the text in the search box or click . To access the list of memorized search strings, click .
	Use this icon to update the list of plugins and their statuses.
	If you have enterprise plugin repositories configured, there is a list that provides repository-based filtering. You can select to see the contents of all repositories o,r only the selected one.
	This list provides category-based filtering. You can select to see all plugins, or the plugins that belong to the selected category.
Sort by	Click and select a category.
Install plugin	Click the Install plugin button in the description area to download and install the selected plugin.
HTTP Proxy Settings	If you access the Internet via an HTTP proxy, click this button and specify the HTTP proxy settings .
Manage repositories	Click this button to create or edit the list of enterprise plugin repositories in the Custom Plugin Repositories dialog .

Context menu commands

Command	Description
Reload List of Plugins	Use this command to update the list of plugins and their statuses.
Sort by | <Category>	Use this command to enable or cancel sorting. The following categories are available: – Status – Downloads – Rating – Last Updated
Download and Install	Use this command to download and install the selected plugin.

Custom Plugin Repositories Dialog

This dialog opens when you click the Manage repositories button in the Browse Repositories dialog .

Use this dialog to manage the list of enterprise plugin repositories . The repositories are identified by their URLs .

Icon	Shortcut	Description
	Alt+Insert	Use this icon or shortcut to add a new repository to the list. Specify the URL of the repository in the dialog that opens. Use the Check Now button to make sure that the specified URL is correct: IntelliJ IDEA will try to connect to the repository.
	Enter	Use this icon or shortcut to edit the selected URL.
	Alt+Delete	Use this icon or shortcut to remove the selected URL from the list.

Version Control

File | Settings | Version Control for Windows and Linux

IntelliJ IDEA | Preferences | Version Control for OS X

Ctrl+Alt+S

The settings under this node allow configuring integration with different version control systems.

Common settings that are applied to the project files regardless of which version control system is used:

- Confirmation
- Background
- Ignored Files
- Issue Navigation
- Changelist Conflicts

The settings for configuring integration with a specific version control system are located under the following nodes:

- GitHub
- CVS
- ClearCase
- Git
- Mercurial
- Perforce
- SourceSafe
- Subversion
- TFS

Specify which version control systems will be used for specific directories, or the entire project.

Item	Description
Directory	This field shows the path to project directories or the project root(s). For projects with Git or Mercurial integration enabled, IntelliJ IDEA scans project directories to check if there are Git /Mercurial repositories that are not controlled by the IDE. If such repositories are found, they are listed here under Unregistered roots and are marked grey. To add an unregistered root, select it in the list and click the Add button +. IntelliJ IDEA also checks if registered roots are valid, i.e. that a Git /Mercurial repository exists at the specified path. If invalid repositories are detected, they are marked with red.
VCS	Select a version control system for the specified directory. The list only displays the version control systems for which the corresponding plugins are enabled (See Managing Plugins).
…	Click this button to open the Version Control Configurations dialog and update the configuration settings for the selected VCS.
+	Click this button to add a directory mapping to the list. The Add VCS Directory Mapping dialog box opens where you can specify the required directory, select a VCS for it, and open the Version Control Configurations dialog box to configure the specified VCS, if necessary.
	Click this button to edit the selected directory mapping. The Edit VCS Directory Mapping dialog box opens where you can update the selected mapping and configure the specified VCS, if necessary.
−	Click this button to remove the selected directory mapping from the list.
Limit history to	Select this check box to specify the number of history rows. If this check box is selected, the text box of history depth, and the spin box become enabled.
Show directories with changed descendants	If this check box is selected, the directories that contain changes, are color-marked. The colors are configurable in the Colors and Fonts pages of the Editor settings (File Status - Have immediate changed children, Have changed descendants).
Store on shelf base revision texts of files under DVCS	Select this check box to have IntelliJ IDEA always shelve the base revisions of files that are under Git or Mercurial version control. By default, IntelliJ IDEA always "remembers" the last commit hash. However, this information is not sufficient if the history has been changed since the last commit as a result of running the Rebase operation. In this case, having a copy of the base revision may help. This check box is relevant only for integration with distributed version control systems, such as Git and Mercurial .
Show changed in last <number> days	Select this check box to have color indication of file status applied during stacktrace analysis and debugging . The names of the files that have been changed within a certain period will be highlighted accordingly. Specify the number of days.
Filter Update Project information by scope	If this option is enabled and a scope is selected, the files that belong to this scope will be marked in bold in the Update Project Info tab of the Version Control Tool Window . If you click the Filter button in the toolbar in the Update Project Info tab, the files will be filtered by scope, i.e. only the files that belong to the selected scope will be displayed. Click the Manage Scopes link to open the Scopes settings dialog and configure a scope.
Commit message right margin (columns)	In this field, specify the number of symbols that can fit into the right margin of the Commit Changes dialog. Select the Wrap when typing reaches right margin option if you want the text to be transferred to the next line when the maximum number of characters has been reached.
Show unversioned files in Commit dialog	Select this option to see newly added files that have not been added to version control yet under the Unversioned Files node in the Commit Changes dialog .

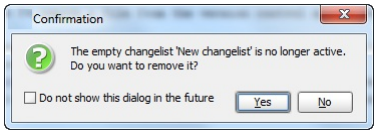
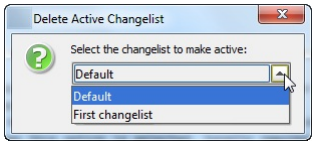
Confirmation

File | Settings | Version Control | Confirmation for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Confirmation for OS X

Ctrl+Alt+S

In this page, specify whether you want IntelliJ IDEA to ask you for confirmation before performing specific version control related actions.

Item	Description
When files are created	In this section, specify whether and how to put a file created in IntelliJ IDEA under version control. The following options are available: – Show options before adding to version control: When this option is selected, newly created files are put under version control after you specify the version control options in the dialog box that opens. – Add silently: When this option is selected, newly created files are automatically put under version control without displaying any messages. – Do not add: When this option is selected, newly created files remain unversioned and you can put them under version control later.
When files are deleted	In this section, specify whether and how to remove a file from the version control system when the file is removed in IntelliJ IDEA. The following options are available: – Show options before removing from version control: When this option is selected, locally removed files are also removed from the specified VCS but first a dialog box for selecting version control options is displayed. – Remove silently: When this option is selected, all locally removed files are removed from the specified VCS without asking for confirmation. – Do not remove: When this option is selected, locally removed files remain under version control.
When empty changelist becomes inactive	In this section, specify IntelliJ IDEA behavior on deleting an empty changelist. – Show options before removing: When this option is selected, IntelliJ IDEA asks for confirmation before removing an empty changelist that has lost its active status:  If you choose to delete such a changelist, IntelliJ IDEA suggests to choose another changelist to be marked as active.  – Remove silently: When this option is selected, IntelliJ IDEA automatically deletes empty changelists that become inactive, except for the Default changelist. – Do not remove: When this option is selected, empty changelist is not deleted on loosing its active status.
Display Option dialogs when these commands are invoked	In this area, specify whether you want IntelliJ IDEA to ask you for confirmation when invoking commands, specified below. To enable showing a prompt before performing an action, select the relevant check boxes. Tip – Availability of commands depends on the particular version control system. – Use controls in this area to restore the original settings, if you have suppressed showing operation-specific option dialog boxes by selecting the Do not show this dialog in the future check box.
Show Clear Read-Only Status dialog box	Select this check box to have IntelliJ IDEA explicitly require cancellation of the read-only status when you open a file in the editor and try to modify it. The read-only status can be cleared in two ways: – Using the current VCS: the file is added to a changelist. – Using a file system: the file is not added to the changelist. This option is relevant for those version control systems that separate check-out from opening for editing.
Suggest to move uncommitted changes to another changelist	When this option is selected, on committing a changelist to the repository with some files excluded from the commit, IntelliJ IDEA prompts to move these files to another changelist.
Force non-empty check-in comments	Select this check box to suppress committing changes without supplying them with corresponding comments.
Clear initial commit message	If this check box is selected, the previous check-in comment is cleared from the Comment area.
Show patch in explorer after creation	Use this drop-down list to define the IntelliJ IDEA behavior when a patch is created . The available options are: – Yes: if this option is selected, native file manager always opens to show the patch file. – No: if this option is selected, the native file manager will not open. – Ask: if this option is selected, IntelliJ IDEA will display a dialog box informing about successful patch creation, and a suggestion to locate the patch file in the native file manager.
Create changelist on failed commit	Use this drop-down list to define the IntelliJ IDEA behavior in case of failed commits. The available options are: – Yes: if this option is selected, the Failed commit changelist will be automatically created, and the respective files will be moved to this changelist. – No: if this option is selected, the files that failed to commit will remain in their original changelist. – Ask: if this option is selected, on failed commit IntelliJ IDEA will display a dialog box asking whether the files should be moved to another changelist, or remain in the original one. See the section Resolving Conflicts .

Background

File | Settings | Version Control | Background for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Background for OS X

Ctrl+Alt+S

In this page, enable performing specific version control related operations in the background without any activities from your side.

Item	Description
Background Operations	In this area, specify the version control related operations to be performed in the background. To enable running an operation in the background, select the relevant check box: – Perform update from VCS in background – Perform commit to VCS in background – Perform checkout from VCS in background – Perform Edit/Checkout in background – Perform Add/Remove in background – Perform revert in background
Changelists to cache initially	Use this spin box to specify the number of changelists to be stored in the cache.
Refresh changes every ... minutes	Use this spin box to specify how often the VCS should check for new changes and refresh the cache. The spin box is only enabled when the Enable background processes check box is selected.
"Changed on server" conflicts	In this area, specify whether you want IntelliJ IDEA to check whether a checked out files have been updated by someone else. To enable background synchronization with the server, select the Check every ... minutes. In the spin box, specify how often you want synchronization to take place. This area is only enabled when the Enable background processes check box is selected.

Ignored Files

File | Settings | Version Control | Ignored Files for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Ignored Files for OS X

`Ctrl+Alt+S`

Use this page to create a list of files and directories that you do not want to put under version control. These can be file names associated with VCS administration, backup files, and any other artifacts that you want to remain unversioned. You can also specify patterns that define files to ignore.

Item	Keyboard shortcut	Description
+	`Alt+Insert`	Use this icon or shortcut to add an item to the list. The Ignore Unversioned Files dialog box opens where you can type an exact path to a file or directory to be ignored or specify a pattern that defines the names of files and directories to be ignored.
	`Enter`	Use this icon or shortcut to edit the selected path or pattern in the Ignore Unversioned Files dialog box.
−	`Alt+Delete`	Use this icon or shortcut to remove the selected path or pattern from the list.

Patterns

Two characters are used as the wildcards:

- `*` : this character matches anything.
- `?` : this character matches a single character.

For example, `*.iml` will ignore all the files with the extension `iml`; `*.?ml` will ignore all the files with the extension ending with `ml`.

Ignore Unversioned Files

File | Settings | Version Control | Ignored Files for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Ignored Files for OS X

`Ctrl+Alt+S`

The dialog box opens when you click the Add + or Edit button in the Ignored Files page.

Use this dialog to configure rules that define which files and folders should be ignored by version control systems. The files you want to ignore can be appointed explicitly by their names of through name patterns with wildcards. To ignore a directory, you need to specify the full path to it relative to the project root.

Select the relevant option and fill in the text box next to it.

Item	Description
Ignore specified file	In this text box, specify the name of the file to be ignored. Do one of the following: – Type the file name relative to the project root, for example, `my_folder/my_subfolder1/my_subfolder2/my_file`. – Click the Browse button … and select the desired file in the Select File to Ignore dialog box.
Ignore all files under	In this text box, specify the name of the directory to be ignored. Do one of the following: – Type the directory name relative to the project root, for example, `my_folder/my_subfolder1/`. – Click the Browse button … and select the desired folder in the Select Directory to Ignore dialog box. The rule is applied recursively to all the subdirectories of the specified directory. If a directory has several subdirectories and you want only one of them ignored, specify the required directory explicitly, for example, `my_folder/my_subfolder1/my_subfolder2/`.
Ignore all files matching	In this text box, specify a pattern that defines the names of files to ignore. The rule is applied to all the directories under the project root. Using wildcards in combination with slashes (`/`) to restrict the scope to a certain directory is not supported.

Issue Navigation

File | Settings | Version Control | Issue Navigation for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Issue Navigation for OS X

Ctrl+Alt+S

Use this dialog to create a list of the so-called issue navigation patterns. An issue navigation pattern maps an issue ID pattern in commit messages with the URL addresses of the referenced issues. This enables you to navigate from committed changes to issues related to these changes. As soon as IntelliJ IDEA encounters a match to the issue ID pattern in a commit message, the match is displayed as a link in the Version Control tool window . If you mention several issues, all of them will be displayed as links. Clicking such link opens the matching issue in the default browser.

Item	Description
Issue	This read-only field shows the issue pattern.
Link	This read-only field shows the link to navigate from the issue pattern in the current row to the issue in the bug tracking system.
	Click this button to create a new issue navigation pattern and link. The Add Issue Navigation Link dialog box opens where you can specify: – A regular expression to define the issue ID. – A regular expression to define the navigation link to the issue. Tip Refer to Regular Expressions Syntax Reference for details on using special characters in regular expressions.
	Click this button to create a new JIRA pattern. The Create JIRA Issue Navigation Pattern dialog box is opened where you can specify the URL to your JIRA installation. The regular expression that defines the pattern is added automatically.
	Click this button to create a new pattern for YouTrack . In the dialog box that opens, specify the URL to your YouTrack installation. The regular expression that defines the pattern is added automatically.
	Click this button to update the selected issue navigation link.
	Click this button to remove the selected issue navigation link from the list.

Example

The example below shows how IntelliJ IDEA applies the abovementioned rules to detect a reference to an issue in a commit message and compose a link to it in the issue tracking system.

Issue ID pattern	The regular expression that defines the format in which issues are referenced in commit messages. `[A-Z]+\-\d+` This regular expressions matches all character strings that consist of two substrings separated by an n-dash character: 1. Substring 1: An unlimited number of upper case alphabetic characters. 2. Substring 2: An unlimited number of digital characters.
Issue link pattern	A combination of the URL address of your issue tracking system and a regular expression that identifies issues in it. `http://mytracker/issue/$0` Here `$0` indicates a back reference to the entire match. This means that as soon as IntelliJ IDEA detects a match in a commit message, it is added to the URL address of the tracker as is.
Matching issue ID	IntelliJ IDEA detects the following reference to an issue in the commit message of interest: `MYPROJECT-110`
Composed issue link	In accordance with the above issue navigation pattern, the detected matching reference is added to the URL of the tracker as is, so the link to the referenced issue is composed as follows: `http://mytracker/issue/MYPROJECT-110`

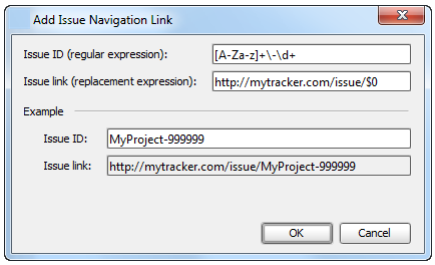
Add Issue Navigation Link Dialog

File | Settings | Version Control | Issue Navigation - Add for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Issue Navigation - Add foc OS X

Use this dialog box to create an issue pattern and navigation link to a bug tracking system.

Item	Description
Issue ID (regular expression)	In this field, type a regular expression that will be converted to the a specific issue id.
Issue link (replacement expression)	In this field, type a regular expression that will be converted to a navigation link to the issue.
Example	In this section, type some specific issue id to make sure it matches the specified pattern:



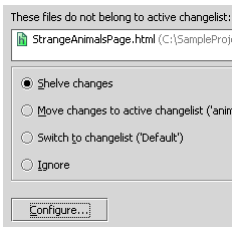
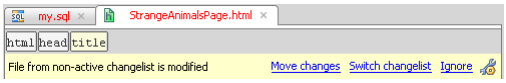
Changelist Conflicts

File | Settings | Version Control | Changelist Conflicts for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Changelist Conflicts for OS X

Ctrl+Alt+S

Use this page to configure protection of files belonging to inactive changelists, from occasional conflicts.

Item	Description
Enable changelist conflict tracking	If this check box is selected, IntelliJ IDEA makes it possible to protect files in inactive changelists. Such protection can be performed on the various levels enabled by the options listed below.
Show conflict resolving dialog	If this check box is selected, IntelliJ IDEA shows Resolve Changelist Conflict dialog box on an attempt to modify a file.  You need to define the way to resolve a conflict, before you proceed with changes. The possible ways to resolve a conflict are as follows: – Shelve changes. – Move file to the active changelist. – Switch changelists to make the current changelist active. – Ignore conflict. In this case, the file in question will be added to the list of files with ignored conflicts.
Highlight files with conflicts	If this check box is selected, IntelliJ IDEA shows a yellow stripe on top of a file from an inactive changelist, when such file has been modified.  In this stripe, you can choose one of the possible ways to resolve conflict: – Move file to the active changelist. – Switch changelists. – Ignore conflict. Names of the files belonging to inactive changelists are shown in red font in the editor tabs and in the Project view.
Highlight files from non-active changelists	If this check box is selected, names of the files belonging to inactive changelists are shown in light-blue font in the editor tabs and in the Project view.
Files with ignored conflicts	This area shows the list of files, for which Ignore option has been selected in the Resolve Changelist Conflict dialog box, or in the editor stripe.
Clear	Click this button to remove all files from the list of files with ignored conflicts.

GitHub

File | Settings | Version Control | GitHub for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | GitHub for OS X

Ctrl+Alt+S

Use this page to specify your GitHub remote storage account credentials or create a GitHub account if you do not have one yet.

Item	Description
Host	Specify the URL of your GitHub repository.
Auth Type	Use this drop-down list to select how you want to be authenticated on GitHub. The available options are: – Password. If this option is selected and you have two-factor authentication enabled in your GitHub account settings, you will be asked to enter an authentication code each time IntelliJ IDEA requires you to log in to your GitHub account. – Token (recommended by GitHub for authentication from third-party applications, as it does not require IntelliJ IDEA to remember your password).
Login	In this text box, type your GitHub logon name. This field is only available if Password is selected as authentication method above.
Password	In this text box, type your GitHub account password. This field is only available if Password is selected as authentication method above.
Test	Click this button to verify the credentials you have specified.
Token	Specify your personal access token. This field is only available if Token is selected as authentication method above.
Create API Token	Click this button if you do not have a personal API token yet. Specify your GitHub credentials in the Login to GitHub dialog that opens and click the Login button. The token will be generated automatically.
Sign up	Click this link to open the Sign up for GitHub page where you can create a GitHub account.
Connection timeout	Specify the time period to wait for connection to be established.

CVS

File | Settings | Version Control | CVS for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | CVS for OS X

`Ctrl+Alt+S`

Use this page to specify the version control settings to be applied to the directories of your project that are under CVS control.

Item	Description
Updating	In this area, define the IntelliJ IDEA behavior when merge conflicts occur during update from the server. – Show dialog - select this option to have IntelliJ IDEA display a dialog box where you can examine, analyze, and resolve possible conflicts before updating. – Skip merging for all project or module files merged with conflicts - select this option to suppress updating files where conflicts occurred during merge. – Get latest repository versions silently - select this option to have your local files in question automatically updated with their latest repository versions.
Use read-only flag for not edited file	Select this check box to have the read-only status assigned to a file automatically after the check out, update, or commit operations.
Show CVS server output	Select this check box to have the server output of CVS commands displayed in the CVS Output tool window.
Default keyword substitution for text files	Use this drop-down list to specify the keyword expansion mode. CVS uses the keyword substitution mode of a file to differentiate binary files from ASCII files and to indicate what type of keyword substitution is applied when files are committed and checked out. The available options are: – keyword&value (`-kkv`) – keyword, value&locker (`-kkvl`) – keyword only (`-kk`) – original string (`-ko`) – binary (`-kb`) – value only (`-kv`)
Global Settings	Click this button to open the Global CVS Settings dialog box.

ClearCase

File | Settings | Version Control | ClearCase for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | ClearCase for OS X

Use this page to specify the version control settings to be applied to the directories of your project that are under ClearCase control.

Item	Description
Work Offline (on edit: hijack instead of checkout)	Select this check box to enable putting your work on another branch (work "offline") and editing a file without checking it out. When this option is selected, IntelliJ IDEA does not check out new files, but simply clears ReadOnly status of the files. The modified files get *Hijacked* status. Note that working offline makes sense in snapshot views only. Performing certain operation offline in dynamic views (for example, renaming or moving) can produce undesirable results.
Reserved Checkouts	Select this check box to specify that only one person at a time can check out an element "reserved".
Check out automatically hijacked file on check in	Select this check box to have IntelliJ IDEA automatically check out "hijacked" files (files that you modified without previously checking out) on check in. When the user checks in a file with the *hijacked* status, the file is treated as follows: – If the check box is selected, the file is reverted. ClearCase saves file content in a temporary location, checks the file out, restores its content, and checks the file in. – If the check box is cleared, IntelliJ IDEA does not process the hijacked files automatically. It is the user's responsibility to apply the Update command and process the hijacked files. You can do the following with a hijacked file: check it out, modify as required, check in changes when ready, and undo the hijack. All actions are available from the Update dialog.
Use UCM model	With this option turned on, IntelliJ IDEA emulates ClearCase' activities with its own changelists. Creating and committing changelists leads to creation of the ClearCase activities and committing into these activities. When working with Base ClearCase views, this option is turned off automatically.
Synchronize activities on refresh	Select this check box to have your activities synchronized on refresh.
Use "-identical" switch during check in	Select this check box to have unchanged files checked in as well.
Restrict history records	Select this check box to restrict the number of stored history records. When showing the history for a file, IntelliJ IDEA retrieves at most the number of records from the history, specified in the text field.
SCR text file	In this text box, specify the fully qualified name of the `scr` file.

Git

File | Settings | Version Control | Git for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Git for OS X

Ctrl+Alt+S

Use this page to specify the version control settings that will be applied to the directories of your project that are under Git control.

Item	Description
Path to Git executable	In this text box, specify the path to the Git executable file. Type the path manually or click the Browse button and specify the path in the dialog that opens.
Test	Click this button to verify the path to the Git executable file.
SSH executable	Use this drop-down list to specify the SSH version to be used with Git. The available options are: – Built-in: select this option to use the implementation provided by IntelliJ IDEA. – Native: select this option to use native implementation. Note that on some platforms, the use of native ssh implementation may cause hang-up issues. You may need to configure a platform-specific `ssh-askpass` to receive prompts for passwords.
Control repositories synchronously	This option only becomes available if you have a multirooted project, i.e. there are several Git repositories within a single project. Select this option if you want branch operations (such as `checkout`, `merge`, etc.) to be applied synchronously to all repositories within your project.
Commit automatically on cherry-pick	When you cherry pick a specific commit, the Commit Changes dialog is displayed. If the Commit automatically on cherry-pick option is selected, the selected commit is submitted silently on clicking the cherry-pick button without displaying the Commit Changes dialog.
Warn if CRLF line separators are about to be committed	Select this option to enable smart handling of `LF` and `CRLF` line separators. IntelliJ IDEA will analyze your configuration, warn you if you are about to commit CRLF into the repository, and suggest changing the `core.autocrlf` setting to `true` or `input` depending on your operating system. Note: Note that this setting is not applied to files where you have set any related Git attributes. In this case, IntelliJ IDEA assumes that you clearly understand what you are doing and excludes such files from analysis. If this option is deselected, you will have to fix issues with line endings manually using the Difference Viewer dialog.
Warn when committing in detached HEAD or during rebase	Select this option if you want IntelliJ IDEA to display a warning when a commit is performed from a detached head or on rebase, as this may cause issues and code loss.
Update method	Use this drop-down list to choose the strategy to synchronize your local repository with the remote storage. The selected method will be used when the `push` operation is rejected (if the Auto-updated if push of the current branch was rejected option is enabled), or when you invoke the Update Project operation. The following options are available: – Merge: choose this option to have the merge strategy applied. The result is identical with that of running `git fetch ; git merge` or `git pull --no-rebase`. – Rebase: choose this option to have the rebase strategy applied. The result is identical with that of running `git fetch ; git rebase` or `git pull --rebase`. – Branch Default: choose this option to have the default command for the branch applied. The default command is specified in the `branch.<name>` section of the `.git/config` configuration file.
Auto-update if push of the current branch was rejected	Select this checkbox if you want the current branch to be updated automatically if the `push` operation from the current branch to its tracked branch is rejected. If this option is deselected, IntelliJ IDEA will display the Push Rejected dialog when pushing a branch is rejected because your local repository and the remote storage are not synchronized. Note: Note the following: – If you have never seen the Push Rejected dialog box before and you are enabling the check box initially, IntelliJ IDEA will update the conflicting local branch silently by means of the `merge` operation. – If you have already encountered the Push Rejected dialog box and selected the Remember the update method choice... option, IntelliJ IDEA saves your last choice (`rebase` or `merge`) and will apply it to update the conflicting local branch silently. Accordingly, to change the "remembered" setting, clear the check box, access the Push Rejected dialog box, select the Auto-update if push ... rejected option, and invoke another update strategy.
Allow force push	If this check box is selected, the Force push option is added to the Git Push dialog (as a drop-down option on the Push button).
Protected branches	If you have selected the Allow force push option, but want to disable it for certain branches, list them here (this is a team-shared parameter that is stored in `.idea/vcs.xml`). You can list several branches separated by a colon, or supply branch patterns as the input is treated as a list of regular expressions.

Mercurial

File | Settings | Version Control | Mercurial for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Mercurial for OS X

`Ctrl+Alt+S`

Use this page to specify the version control settings to be applied to the directories of your project that are under Mercurial control.

Item	Description
Path to hg executable	Specify the location of the Mercurial executable file. Enter the path manually, or click the Browse button and select the path in the dialog that opens. If you followed the standard installation procedure, the default location is `/opt/local/bin` or `/usr/local/bin` for Linux and OS X and `/Program Files/TortoiseHG` for Windows. It is recommended that you add the path to the Mercurial executable file to the `PATH` variable. In this case, you can specify only the executable name, the full path to the executable location is not required. For more information about environment variables, see Path Variables .
Test	Click this button to verify the path to the Mercurial executable.
Control repositories synchronously	This option only becomes available if you have a multirooted project, i.e. there are several Mercurial repositories within a single project. Select this option if you want branch operations (such as `checkout`, `merge`, etc.) to be applied synchronously to all repositories within your project.
Check for incoming and outgoing changesets	Select this option if you want IntelliJ IDEA to detect incoming and outgoing changes in the background mode. IntelliJ IDEA will automatically request the server for incoming and outgoing changesets every 5 minutes.
Ignore whitespace differences in annotations	Select this check box if you want white spaces to be ignored when annotating, and, thus, get more meaningful annotations and cast out senseless ones.

Perforce

File | Settings | Version Control | Perforce for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Perforce for OS X

`Ctrl+Alt+S`

Use this page to specify the version control settings to be applied to those directories of your project that are under Perforce control

Item	Description
Perforce is online	Select this check box to work with Perforce in the online mode.
Switch to offline mode automatically if Perforce is unavailable	Select this check box to have IntelliJ IDEA automatically go offline as soon as Perforce becomes unavailable and display the corresponding message.
Use P4CONFIG or default connection	If this option is selected, `P4CONFIG` or the default Perforce connection are used to connect to the Perforce server. Using `P4CONFIG` makes it trivial to switch between Perforce settings for different projects, when necessary.
Use connection parameters	If this option is selected, the connection credentials (port, client, user name, and charset) are specified manually.
Port	In this text box, type the server and the port which your Perforce client will listen to. For the default Perforce server configuration, it looks like *perforce:1666*.
Client	In this text box, type the name of your Perforce workspace name.
User	In this text box, type your user name to authenticate to the server.
Charset	From this drop-down list, select the character set to be used.
Dump Perforce commands to <path>	Select this check box to have IntelliJ IDEA create a file `P4.output` and store the output of Perforce commands in it.
Use login authentication	When this check box is selected, Perforce requires a login to authenticate a user.
Test Connection	Click this button to check whether the specified settings ensure establishing connection to the Perforce server.
Path to P4 executable	In this text box, specify the path to the Perforce Command Line Client's executable file `P4` . Click the Browse button ... to open the Select Path - P4 Configuration dialog box and select the executable file in the directories tree.
Path to P4V executable	In this text box, specify the path to the Perforce Visual Client's executable file `P4V` . Click the Browse button ... to open the Select Path - P4 Configuration dialog box and select the executable file in the directories tree.
Show branching history ...	Select this check box to enable displaying the branch history of a specified file, including all file branch points, edits, and merges.
Show integrated changelists in committed changes	Select this check box to have IntelliJ IDEA point at committed changes that are also integrated to other changelists and provide information on the target changelists that received the content in question.
Server timeout	In this text box, specify the time period in seconds after when the Perforce client cancels its attempts to establish connection to the server.
Enable Perforce Jobs Support	When this check box is selected, user interface for attaching and detaching Perforce jobs to change lists is provided in the Version Control tool window and in the Commit Changes dialog box.

SourceSafe

File | Settings | Version Control | SourceSafe for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | SourceSafe for OS X

`Ctrl+Alt+S`

Use this page to specify the version control settings to be applied to those directories of your project that are under Visual SourceSafe control

Item	Description
Path to VSS client (ss.exe)	In this text box, specify the location of the SourceSafe client executable file `ss.exe`.
	Click this button to access the Open dialog box for choosing the `ss.exe` file.
Path to VSS configuration file (srcsafe.ini)	In this text box, specify the location of the SourceSafe configuration file `srcsafe.ini`.
	Click this button to access the Open dialog box for choosing the `srcsafe.ini` file.
User name	In this text box, type your VSS user ID for accessing the repository.
Password	In this text box, type your VSS user password.

Subversion

File | Settings | Version Control | Subversion for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Subversion for OS X

Ctrl+Alt+S

Use this page to specify the version control settings to be applied to the directories of your project that are under Subversion control

In this section:

– General tab
– Presentation tab
– Network tab
– SSH Settings

General tab

Use this tab to configure the general Subversion integration settings.

Item	Description
Use command line client	Select this option if you want to use the command line svn client. Enter the name of the executable file, or click the Browse button ... and select the path in the dialog that opens.
Enable interactive mode	Select this option if you want IntelliJ IDEA to emulate the behavior when Subversion commands are executed directly from the terminal in the interactive mode (dialogs will pop up where you can input credentials). This is required to handle password/passphrase prompts for svn+ssh repositories, and trust invalid server certificates for https repositories.
Use system default Subversion configuration directory	Select this check box to store Subversion configuration files in the system default location.
Subversion configuration directory	In this text box, specify the Subversion configuration directory if you do not want to use the default one. Enter the path manually, or click the Browse button ... and select the path in the dialog that opens. This option is only available when the Use system default Subversion configuration directory option is deselected.
Update administrative information only in changed subtrees	This option only applies to working copies older than SVN 1.7 managed by SVNKit. During synchronization with the server (update), SVN locks your working copy one subtree after another by creating empty `lock` files in the corresponding administrative `.svn` directories. After that, SVN starts comparing file hashes to detect which local files need to be synchronized. When this option is selected, SVN first checks if any files from a subtree have been modified on the server, and locks this subtree (i.e. creates a `.svn/lock` file) only if such files are detected. This approach improves performance but may cause concurrency issues, for example, with antiviral software.
Clear Auth Cache	Click this button to delete all stored credentials for the `http`, `svn` and `svn+ssh` protocols from the authentication cache.

Presentation tab

Use this tab to configure the data presentation settings.

Item	Description
Check svn:mergeinfo in target subtree when preparing for merge	Select this check box if you want IntelliJ IDEA to check the merge tracking information for the target branch before merging to prevent duplicates.
Maximum number of revisions to look back in annotations	Select this check box to limit the number of revisions to look back at when calculating annotations, and specify the number of revisions.
Show merge source in history and annotations	Select this check box if you want merge sources to be visible in annotations and file history.
Ignore whitespace differences in annotations	Select this check box if you want white spaces to be ignored when annotating, and, thus, get more meaningful annotations and cast out senseless ones.

Network tab

Use this tab to configure the connection settings.

Item	Description
Use IntelliJ IDEA general proxy settings as default for Subversion	Select this check box if you want Subversion to use the default IntelliJ IDEA proxy settings.
HTTP timeout	Specify the number of seconds to wait for HTTP connection to be established.
SSH connection timeout	Specify the number of seconds to wait for SSH connection to be established.
SSH read timeout	Specify the number of seconds to wait for response.
SSL protocols	In this area, select which SSL protocol you want to use. The available options are: – All – SSLv3 – TLSv1
Edit Network Options	Click this button to change Subversion runtime configuration file in the Edit Subversion options related to the network layers dialog .

SSH Settings

Use this tab to configure the settings used to connect to an SVN server via a tunneling SSH protocol.

Item	Description
SSH executable	Specify the path to the SSH client. Enter the name of the executable file, or click the Browse button … and select the path in the dialog that opens. If not specified, `'ssh'` is used by default. This field is only available if the Password or the Private key option is selected.
User name	Specify the user name for SSH connection. If the user name is explicitly specified in the repository URL, this value will be used and this setting will be ignored. This field is only available if the Password or the Private key option is selected.
Port	If your server is listening on a non-standard port (22 for svn+ssh://), modify the default value. This field is only available if the Password or the Private key option is selected.
Password	Select this option if you want to use a password for SSH authentication.
Private key	Select this option if you want to use a private key for SSH authentication.
Path	Specify the path to the private key. Enter the path manually, or click the Browse button … and select the path in the dialog that opens.
Subversion config	Select this option if you want to use the default settings stored in Subversion configuration for SSH connection.
SSH tunnel	This field displays the SSH tunnel settings stored in Subversion configuration. You can modify the value and click the Update button to write this value to the Subversion configuration.
Update	Click this button to check the Subversion configuration and update the value if necessary, or to write the value you have entered to the Subversion configuration.
SVN_SSH	This field displays the environment variable that can be used in the tunnel configuration (by default, `SVN_SSH`) and is stored in Subversion configuration.

Edit Subversion Options Related to Network Layers Dialog

File | Settings | Version Control | Subversion

The dialog box opens when you click the Edit Network Options button on the Subversion page of the Settings dialog box. In this dialog box, specify the Subversion network settings stored in the servers Subversion runtime configuration file.

The dialog box contains two tabs:

- System file - this tab displays the default network configuration settings specified by the system administrator.
- User file - in this tab, customize the default network configuration settings.

The dialog box consists of two panes:

- On the left-hand pane, add, edit, and remove configuration profiles. Network configuration settings are arranged into profiles of two types:
  - Group - settings from such profile apply to a specific group, defined by a glob pattern.
  - Global - settings from this profile are applied to all servers that do not match any glob pattern.
- On the right-hand pane, specify the settings for the selected configuration profile.

Toolbar Options

Item	Tooltip and shortcut	Description
	Add `Insert`	Click this button to have a new configuration profile added to the list.
	Delete `Delete`	Click this button to remove the selected profile from the list.
	Copy `Ctrl+C`	Click this button to have a copy of the selected profile created.

HTTP Proxy Settings

Item	Description
URL Patterns	In this text box, type the patterns that define the URL addresses of repositories to be accessed via proxy. Use commas to separate patterns.
Exceptions	In this text box, type the patterns that define the URL addresses of repositories to be accessed directly, without using proxy. Use commas to separate patterns.
Server	In this text box, specify the name or IP address of the proxy server to use.
Port	In this text box, specify the port number the proxy server listens to.
Connection timeout	In this text box, specify the time period in seconds after when the Subversion client cancels its attempts to establish connection to the server.
User	In this text box, type the user name or login to authenticate at the specified proxy server.
Password	In this text box, type the password that matches the specified login or user name.

SSL Settings

Item	Description
Comma separated paths to CAs certificate files	In this text box, specify the paths to files that contain certificates of the Certificate Authority (CAs) files that are accepted by the Subversion client when accessing the repository.
SSL client certificate file	In this text box, specify the location of the SSL client certificate file. Type the path to the file manually or click the Browse button and choose the location in the dialog that opens .
SSL client certificate passphrase	In this text box, type the SSL client certificate passphrase to use.
Trust default CAs	Select this check box to have the Subversion integration trust the set of default Certificate Authority files shipped with OpenSSL .

Repositories

Item	Description
Repositories	This list displays the URL addresses of the previously accessed repositories.
Test connection	Click this button to make sure that connection to the selected repository can be established successfully according to the settings specified in the dialog box. Tip When you click this button, IntelliJ IDEA displays the Authentication Required dialog box.

TFS

File | Settings | Version Control | TFS for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | TFS for OS X

Ctrl+Alt+S

Use this page to specify the version control settings to be applied to those directories of your project that are under TFS control.

Item	Description
Servers and workspaces	In this area, configure access to workspaces and servers to use. – Manage: click this button to open the Manage TFS Servers and Workspaces dialog box where you can create a list of servers and workspaces you need to have access to. – Use IntelliJ IDEA HTTP Proxy settings for TFS: when this check box is selected, TFS servers are accessed through the Proxy server using the IntelliJ IDEA default Proxy settings .
Passwords	In this area, handle the passwords for accessing TFS servers and workspaces. – Reset Saved Passwords: click this button to discard the stored passwords.
Check-in policies compatibility	A check-in policy is a rule that is executed before every check-in to ensure that the selected changeset is OK to commit. Standard policies are stored on the server and are executed on the client machines. Custom policies are implemented as custom plugins to IntelliJ IDEA. The IDs of these plugins are stored on the server, while the policies themselves are applied locally. Therefore, to enable the use of a policy in a team, all the team members should install the corresponding plugin. Use the controls in this area to configure how IntelliJ IDEA should treat third party check-in policies. These settings are applied to all IntelliJ IDEA projects by default unless they are overridden for a specific project. – Evaluate Team Explorer policies: select this check box to have the Microsoft Team Explorer policy definitions installed and executed on the client machine. – Evaluate Teamprise policies: select this check box to have the Teamprise policy definitions installed and executed on the client machine. – Warn about not installed policies: select this check box to have warnings displayed in case the specified policy definition is not installed.

Manage TFS Servers and Workspaces

File | Settings | Version Control | TFS for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | TFS for OS X

Ctrl+Alt+S

The dialog box opens when you click the Manage servers and workspaces button in the TFS dialog box. Use this dialog box to handle the list of TFS servers and workspaces you have access to.

Item	Description
Server/Workspace	This read-only field shows the URL addresses of TFS servers you have access to and workspaces available on these TFS servers.
Workspace comment	This read-only field shows the descriptions of workspaces on the servers you have access to.
Team Servers	Use the buttons in this area to manage the list of available servers and workspaces and configure access to them. – Add: click this button to open the Add Team Foundation Server dialog box where you can specify the parameters for establishing connection to a TFS server. TFS uses NTLM authentication , so native Windows applications (that is, Microsoft Team Explorer) authenticate silently with system credentials. IntelliJ IDEA users must always specify their username and password because of limitations posed by Java Runtime. – Remove: click this button to remove the selected server from the list. – Reload workspaces: click this button to have the list of available workspaces refreshed. – TFS Proxy: click this button to open the Set TFS proxy for server... dialog box where you can specify the parameters for accessing the selected server via Proxy. – Check-in Policies: click this button to open the Edit Check-in Policies dialog box where you can manage the list of check-in policies to be applied.
Workspaces	Use the buttons in this area manage the list of available workspaces and update the workspaces, when applicable. – Create: click this button to open the Create Workspace dialog box for creating a new workspace. – Edit: click this button to open the Edit Workspace dialog box for editing the selected workspace. – Delete: click this button to remove the selected workspace from the list.
Close	Click this button to save the settings, close the dialog box, and return to the TFS dialog box.

Add Team Foundation Server

File | Settings | Version Control | TFS for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | TFS for OS X

Ctrl+Alt+S

The dialog box opens when you click the Add button in the Manage TFS Servers and Workspaces dialog box. Use this dialog box to specify and edit the parameters for establishing connection to TFS servers.

Item	Description
Address	In this text box, type the URL address of the TFS server you want to connect to.
Auth	In this field, specify the authentication protocol to access the server. TFS uses NTLM authentication , so native Windows applications (that is, Microsoft Team Explorer) authenticate silently with system credentials. IntelliJ IDEA users must always specify their username and password because of limitations posed by Java Runtime. To authenticate through OAuth (Windows Live ID), choose Alternate from the Auth drop-down list.
User name	In this text box, type your TFS user name. The field is available for the NTLM) and Alternate authentication types.
Domain	In this text box, type the name of the network domain where the TFS server is located. The field is available for the NTLM) authentication type.
Password	In this text box, type your TFS password. The field is available for the NTLM) and Alternate authentication types.
Store password	Select this check box to have IntelliJ IDEA remember the specified password.

Edit Check-in Policies Dialog

File | Settings | Version Control | TFS for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | TFS for OS X

Ctrl+Alt+S

The dialog box opens when you select an entry in the Server/Workspace list and click the Check-in Policies button in the Manage TFS Servers and Workspaces dialog box.

A check-in policy is a rule that is executed before every check-in to ensure that the selected changeset is OK to commit. Standard policies are stored on the server and are executed on the client machines.

Custom policies are implemented as custom plugins to IntelliJ IDEA. The IDs of these plugins are stored on the server, while the policies themselves are applied locally. Therefore, to enable the use of a policy in a team, all the team members should install the corresponding plugin.

Use this dialog box to manage the list of the custom project policies to be applied when checking in to the selected workspace and to override the default IntelliJ IDEA-wide policies for the project, if necessary.

In this section:

– Check-in Policies
– Compatibility

Check-in Policies

Item	Description
Team Project	From this drop-down list, select the name of the project to specify the policies for.
Policy Type	This read-only field shows the available policies.
Description	This read-only field shows brief descriptions of policies.
Enabled	When this check box is selected, the policy next to it is mandatory during check-in.
Add	Click this button to open the Add Check-in Policy dialog box where you can define a new check-in policy.
Edit	Click this button to open the Edit Check-in Policy dialog box where you can re-define the selected check-in policy.
Remove	Click this button to remove the selected check-in policy from the list.

Compatibility

Use the controls in this area to suppress applying the default IntelliJ IDEA-wide check-in policy settings to the current project.

Item	Description
Override default settings for team project <project name>	Select the check box to discard the default policy settings within the scope of the current project and re-define the settings by selecting or clearing the corresponding check boxes below. – Evaluate Team Explorer policies: select this check box to have the Microsoft Team Explorer policy definitions installed and executed on the client machine. – Evaluate Teamprise policies: select this check box to have the Teamprise policy definitions installed and executed on the client machine. – Warn about not installed policies: select this check box to have warnings displayed in case the specified policy definition is not installed.

Create Workspace

File | Settings | Version Control | TFS for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | TFS for OS X

Ctrl+Alt+S

The dialog box opens when you click the Create button or select a workspace and click the Edit button in the Manage TFS Servers and Workspaces dialog box.

Use this dialog box to define a new workspace or update the existing one by selecting the necessary folders on the TFS server and mapping them to local folders.

Item	Description
Name	In this text box, specify the name of the new workspace.
Comment	In this text box, describe briefly what this workspace is intended for.
Server	This read-only field displays the URL address of the TFS server on which the new workspace will be created and which is selected in the Manage TFS Servers and Workspaces dialog box.
Owner	This read-only field displays your TFS user name.
Computer	This read-only field displays the name of your computer in the network domain.

Working Folders

In this area, define mappings between the necessary folders on the TFS server and local folders.

Tip You can map a folder including all its subfolders recursively or map each subfolder separately.

Item	Description
Status	From this drop-down list, select the status of a new mapping.
Local path	In this text box, specify the path to the local folder. Use the ... button, if necessary.
Server path	In this text box, specify the path to the corresponding folder on the server.
Add	Click this button to create a new mapping.
Remove	Click this button to remove the selected mapping from the list.

Build, Execution, Deployment

File | Settings | Build, Execution, Deployment

When you select the Build, Execution, Deployment category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- Build Tools
- Compiler
- Debugger
- Deployment
- Arquillian Containers
- Application Servers
- Clouds
- Coverage
- Docker Registry
- Required Plugins
- Web Services

Build Tools

File | Settings | Build, Execution, Deployment | Build Tools

When you select the Build Tools category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- SBT
- Maven
- Gradle
- Gant

SBT

File | Settings | Build, Execution, Deployment | Build Tools | SBT

Use this page to configure SBT project settings.

Item	Description
Linked SBT projects	This area shows all linked projects.
Project-level settings	Use this are to configure the following options: – Use auto-import - select this check box to resolve all the changes made to the SBT project automatically every time you refresh your project. – Create directories for empty content roots automatically - select this option to add the *src* directory to your project. – Download sources and docs - select this check box to download sources and docs for project dependencies. – Download SBT sources and docs - select this check box to download sources docs for SBT itself.
JVM	Use this area to configure the JVM settings. You can choose from the following options: – From project JDK - select this default option to use the project's JDK. – Custom - select this option to use a custom JVM.
JVM Options	Use this area to configure additional JVM settings. You can choose from the following options: – Maximum heap size, MB - use this field to specify the maximum heap size available to the process that launches the compiler. The default 768 Mb is suitable for most of the purposes. – VM parameters - use this field to type the string to be passed to the VM when IntelliJ IDEA launches the compiler. If you need more room to type, click + to open the VM parameters dialog where the text entry area is larger.
Launcher (sbt-launch.jar)	Use this area to configure settings for the sbt launcher. You can select from the following options: – Bundled - use this default option if you want the bundled launcher. – Custom - use this option to specify a custom launcher.

Maven

File | Settings | Build, Execution, Deployment | Build Tools | Maven

Item	Description
Work offline	If this check box is selected, Maven works in offline mode and uses only those resources that are available locally. This option corresponds to the `--offline` command line option.
Use plugin registry	Select this check box to enable referring to the Maven's Plugin Registry . This option corresponds to the `--no-plugin-registry` command line option.
Execute goals recursively	If this check box is selected, the build recurses into the nested projects. Clearing this check box corresponds to the `--non-recursive` command line option.
Print exception stack traces	If this option is checked, exception stack traces are generated. This option corresponds to the `--errors` command line option.
Always update snapshots	Select this checkbox, if you want IntelliJ IDEA to update snapshots on sync.
Output level	Select the desired level of the output log, which allows plugins to create messages at levels of *debug*, *info*, *warn*, and *error*, *fatal*, or disable output log.
Checksum policy	Select the desired level of checksum matching while downloading artifacts. You can opt to fails downloading, when checksums do not match (`--strict-checksums`), or issue a warning (`--lax-checksums`).
Multiproject build fail policy	Specify how to treat a failure in a multiproject build. You can opt to fail the build: – At the very first failure, which corresponds to the command line option `--fail-fast` . – Fail at the end, which corresponds to the command line option `--fail-at-end` . – Ignore failures, which corresponds to the command line option `--fail-never` .
Plugin update policy	Select plugin update policy from the drop-down list. You can opt to: – Check for updates, which corresponds to the command line option `--check-plugin-updates` . – Suppress checking for updates, which corresponds to the command line option `--no-plugin-updates` . This option is ignored for Maven 3 and later versions.
Threads `(-T option)`	Use this field to set the `-T` option for parallel builds. This option is available for Maven 3 and later versions. For more information, see parallel builds in Maven 3 feature.
Maven home directory	Use this drop-down list to select a bundled Maven version that is available (for Maven2, version 2.2.1 and for Maven3, version 3.0.5) or the result of resolved system variables such as `MAVEN_HOME` or `MAVEN2_HOME` . You can also specify your own Maven version that is installed on your machine. You can click … and select the necessary directory in the dialog that opens .
User settings file	Specify the file that contains user-specific configuration for Maven in the text field. If you need to specify another file, check Override option, click ellipsis button and select the desired file in the Select Maven Settings File dialog.
Local repository	By default, the field shows the path to the local directory under the user home, that stores the downloads, and contains the temporary build artifacts that you have not yet released. If you need to specify another directory, check Override option, click ellipsis button and select the desired path in the Select Maven Local Repository dialog.

Maven. Importing

Item	Description
Keep projects files in	Select this check box to specify the location of your project's files after the import. For example, when you import a project and want to keep the `.iml` file and `.idea` directory files in a specific location instead of the default one. By default, IntelliJ IDEA places project's files next to your `pom.xml`.
Import Maven projects automatically	Select this check box, if you want IntelliJ IDEA to perform reimport automatically each time you change your `pom.xml`.
Create IntelliJ IDEA modules for aggregator projects (with 'pom' packaging)	If this check box is selected, IntelliJ IDEA Maven Modules will be created for each module included in the `pom.xml` file of an aggregative project, provided that its packaging is set to 'pom'.
Create module groups for multi-module Maven projects	If this check box is selected, IntelliJ IDEA will create a module group from an aggregative Maven project, with the nested modules included in this group.
Keep source and test folders on reimport	If this check box is selected, all the source and test folders will be preserved on every import. If this check box is cleared, all previously configured source and test folders will be removed on every import. By default, this check box is set as follows: – For new projects: the check box is cleared. – For already imported projects: the check box is selected.
Exclude build directory (`PROJECT_ROOT/target`)	Select this check box to exclude a build directory from the project. This might be useful, if you want to speed up the project's importing process . If this check box is cleared, IntelliJ IDEA will index files in the build directory every time you import a project which might take additional time.
Use Maven output directories	If this check box is not selected, the build will be created in the regular IntelliJ IDEA's output directory `USER_HOME\IdeaProjects\<project>\classes\Production\`. If this check box is selected, the build is generated in the Maven's output directory, and the results of IntelliJ IDEA's compilation are reused. However, IntelliJ IDEA itself does not reuse Maven build results, and performs compilation *from scratch*.
Generated sources folders	Specify the directory of your source root when you reimport a project. You can select one of the following options: – Detect automatically This is a default option. When you select this option, IntelliJ IDEA automatically detects the location of the generated sources. IntelliJ IDEA also detects which directory to mark as a source root. However, IntelliJ IDEA searches for the generated sources only in target/generated-sources and target/generated-sources/* directories. – target/generated-sources This option enables you to mark the directory as source root manually. – subdirectories of "target/generated-sources" This option enables you to mark a subdirectory as a source root manually. – Don't detect This option lets you skip the detection process.
Phase to be used for folders update	Select Maven phase to be used for folders update. This might be useful, if you adjust your plugins so that additional sources are loaded at some phase.
Automatically download	Select the corresponding check boxes to automatically download sources (Sources) and documentation comments (Documentation) on opening Maven projects.
Dependency types	Use this field to specify dependency types that you want to include when you reimport your project.
VM options for importer	Use this field to specify VM options. The default option is `-Xmx512m`. When you specify the options, follow the following rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`.
JDK for importer	Use this drop-down list to specify which JDK to use when the maven project is imported. You can choose one of the following options: – Internal JRE - this is a default option that uses the installation directory for JRE. – JDK versions - this option uses your project's JDK. – Use JAVA_HOME - this option uses the value that is specified in the user's environment variable settings.
Generate Flex compiler configuration files when importing Flexmojos projects	If this check box is selected, when importing Flexmojos projects, the Flex compiler configuration files are generated automatically. The automatic generation of the configuration files is a rather time-consuming process, especially for large projects. Besides, for the reasons independent of IntelliJ IDEA, the automatic generation of the configuration files may sometimes be impossible or may lead to erroneous results. In all such cases, you may want to turn this option off and generate the Flex compiler configuration files from the command line using this command: `mvn compile -DconfigurationReport=true` Note For this option to be available, the Flash/Flex Support enabled in IntelliJ IDEA. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins.
Create run configuration for Spring Boot application	Select this option if you want IntelliJ IDEA to automatically create a run/debug configuration for Spring Boots projects on Maven import.

Maven. Ignored Files

Use this page to specify the file types and individual files, which should not be included in build.

Item	Description
Path patterns	Enter the comma-separated list of paths to be ignored during the build. Wildcards are honored.
Ignored files	Check the individual files to be ignored during the build.

Maven. Runner

Use this page to configure settings for the external Maven that will be used to run goals.

Item	Description
Run in background	Check this option to perform run as a background task.
VM Options	Specify VM options that will be passed to the selected JRE. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`.
JRE	Select the JRE that will be used to run the Maven goals.
Environment variables	This field lets you set custom environment variables on the project level for running maven goals. Click the Browse button to open the Environment Variables dialog box, where you can create variables and specify their values.
Properties	Specify the properties and their values to be passed to Maven.
(`Alt+Insert`)	Use this icon or shortcut to define a new property as a name - value pair.
(`Enter`)	Use this icon or shortcut to change the selected property.
(`Alt+Delete`)	Use this icon or shortcut to remove the selected properties from the list.
Skip tests	Check this option to skip performing unit tests. This corresponds to the Maven option `-Dmaven.test.skip=true`

Maven. Running Tests

Use this page to configure settings for running JUnit tests using Maven Surefire plugin configuration.

The configuration parameters are activated by default.

You can use these settings for Maven 2 and Maven 3 versions.

Item	Description
`argLine`	Select this check box to set arbitrary JVM options on the command line.
`systemPropertyVariables`	Select this check box to pass a list of System properties to the JUnit tests.
`environmentVariables`	Use this check box to set additional environment variables on the command line.

Maven. Repositories

The table shows the list of Maven repositories , encountered in the current project, with their URLs, type (local or remote) and the date of the most recent update.

Item	Description
Indexed Maven Repositories	This area contains a list of Maven local repositories that are configured in the `pom.xml` file. The list is updated automatically. If you open a project that contains additional repositories specified, then the repositories are added to the Indexed Maven Repositories list and you can update the indices.
Update	Click this button to update indices of the selected repository. It might be helpful in case you expect to get information for newly deployed artifacts such as new versions of libraries that you use in the project. Also when you use maven dependencies completion in `pom.xml` or generation of maven dependencies using Maven Artifact Search dialog.
Artifactory or Nexus Service URLs	This area contains a list of remote repositories.
Add	Click this button to add a new URL to the list.
Edit	Click this button to changes the selected URL.
Remove	Click this button to remove the selected URL from the list.
Test	Click this button to test connection, scan the selected URL and discover the available repositories. The list of the repositories found at the specified URL is shown in a pop-up frame.

Gradle

File | Settings | Build, Execution, Deployment | Build Tools | Gradle

Use this page to configure Gradle project settings.

Item	Description
Linked Gradle projects	This area contains the list of registered Gradle projects that are linked to your IntelliJ IDEA project.

Project-level settings	This area contains settings for your Gradle project. You can select from the following options:
	– Use auto-import - select this check box to resolve all the changes made to the Gradle project automatically every time you refresh your project.
	– Create directories for empty content roots automatically - select this option to add a `src` directory to your project automatically when you import a project from Gradle model.
	– Create separate module per source set - select this check box to use the source set feature in resolving your Gradle projects.
	– Use default gradle wrapper (recommended) - select this check box to use Gradle Wrapper. You can use this option when you have generated or checked out wrapper files in the default location.
	`gradle/wrapper/gradle-wrapper.jar (Wrapper JAR)`
	`gradle/wrapper/gradle-wrapper.properties (Wrapper properties)`
	– Use gradle wrapper task configuration - select this check box to customize your Gradle Wrapper. You can use this option if you do not have wrapper files on your disk yet or if you use a different location for them. In this case IntelliJ IDEA refers to the Gradle wrapper task definition and generates or updates the files based on the task configuration. This option can be useful when you don't want to check in binary wrapper files or reuse the same wrapper files for several projects.
	`task wrapper(type: Wrapper) {`
	`  distributionUrl = "http://mycompanyserver/gradle-2.10-bin.zip"`
	`  jarFile = "/mylocation/gradle/wrapper/gradle-wrapper.jar"`
	`  scriptFile = "/mylocation/gradle/gradlew"`
	`}`
	Note This option is supported for the Gradle version 1.7 or later.
	– Use local gradle distribution - select this option to run local build scripts.
	– Gradle home - in this text field, specify the fully qualified path to your Gradle installation.
	If Gradle location has been defined by the environment variables `GRADLE_HOME` or `PATH`, then IntelliJ IDEA deduces this location, and suggests this path as the default value.
	
	If Gradle location has not been deduced from the environment variables, specify it manually, or click the Browse button, and select the desired directory in the dialog that opens. Note that the value entered in this field takes preference over the environment variables.
	– Gradle JVM - use this drop-down list to select a JVM for running Gradle projects. The default is set to your project JDK.

Global Gradle settings	This area contains options for Gradle global settings. You can select from the following options: – Offline work - use this check box to work with Gradle in the offline mode. In this case Gradle will use dependencies from the cache. Gradle will not attempt to access the network to perform dependency resolution. If required dependencies are not present in the dependencies' cache, a build execution will fail. – Service directory path - use this field to override the default Gradle home location directory. – Gradle VM options - use this field to specify VM options for your Gradle project. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m`. – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`.

Runner

IntelliJ IDEA lets you specify settings for running Gradle tests.

Item	Description
Run tests using	Use this area to select one of the following test runner options: – Platform Test Runner - select this option if you want to run your tests using the IntelliJ IDEA API. – Gradle Test Runner - select this option if you want to run your tests using Gradle tooling API . – Let me choose per test - select this option if you want to choose the runner for the specific test. In this case IntelliJ IDEA prompts you to select the test runner the first time you try to execute your test.

Experimental

This page appears when you have an Android project and work inside the Android environment.

Item	Description
Allow Module selection on Project import	Select this check box to allow a partial import.
Skip source generation on Gradle sync if a Project has more than <number of modules> Modules	Select this check box to skip source generation if your project contains more than the specified number of modules. It might be helpful if you have a multi-module project.

Gant

File | Settings | Build, Execution, Deployment | Build Tools | Gant

Use this page to define Gant home directory, which is required to be able to run build scripts.

Item	Description
Gant home	In this text field, specify the fully qualified path to your Gant installation, or click the ellipsis button, and select the desired directory in the dialog that opens .

Compiler

File | Settings | Build, Execution, Deployment | Compiler for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler for OS X

Use this node to configure common options specified in the table below, as well as the specific options for compilers used in IntelliJ IDEA.

Item	Description
Resource Patterns	In this field, specify the regular expression that describes the files that should be recognized as resources and, consequently, copied to the output directory. Use semicolons (`;`) to separate individual patterns. Wildcards and negations are welcome. The following symbols are accepted: – `*` represents an unlimited number of any symbols, possibly none. – `?` represents exactly one symbol. – `.` represents a delimiter. – `!` negates the entire mask it is applied to. Consequently, any file with the name and extension that do not match the pattern will be recognized as a resource file. – `/` represents a path separator. – `/**/` denotes any number of directories. – `<dir>:<pattern>` denotes any directory located under the source root `<dir>` ; `<pattern>` is any pattern that meets the above-mentioned requirements. The examples below illustrate the use of wildcards in the resource patterns: – `*.xml` - any XML file. – `!*.xml` - any file whose extension is not `.xml` . – `z*.properties;z*.gif;z*.png;z*.jpeg;z*.xml` - any `.properties` , `.gif` , `.png` , `.jpeg` , or `.xml` file with the name beginning with `z` . – `MyResources:*` - all files and folders within the directory `MyResources` . Tip If you want to skip compilation of certain Groovy files in the modules with the Groovy support, include them in the list of the resource patterns.
Clear output directory on rebuild	Check this option to delete all files in the output directories. Do not check this option, if the output directory contains files IntelliJ IDEA is not aware of, like resources, etc. If there is any intersection of source and output paths, you will be prompted to resolve the issue by separating source and output directories, or ignore the issue.
Add @NotNull assertions	If this option is checked, the `@NotNull` assertions are added at run-time.
Automatically show first error in editor	If this check box is selected, the file that contains the very first compilation error will be opened in the editor, with the highlighted line that contains the error.
Make project automatically	Select this check box to automatically make (compile) the project each time project files change on your disk, for example, on save or autosave, or when you get the latest project revision from your version control system.
Compile independent modules in parallel	If this check box is selected, the modules without mutual dependencies are compiled simultaneously. This might require increased heap size .
Rebuild modules on dependency change	Select this check box to have the modules with the changed dependencies fully rebuilt.
Build process heap size (Mbytes)	In the text field, specify the heap size required for the build process. Note – If you are using a 64-bit JDK for compilation, the build process may require more memory. – The value is stored with the project settings. If you need to override this value, then in the field User-local build process VM options write `Xmx<N>m` , where `<N>` is the heap size value in megabytes. As soon as this value is recognized in the field User-local build process VM options, the field Build process heap size becomes read-only and is ignored.
Shared build process VM options	These VM options will be added to the command line on launching the build process. The shared VM options are stored in the project settings and may be put under version control .
User-local build process VM options (overrides Shared options)	These VM options will be added to the command line on launching the build process. The user-local VM options are stored in `workspace.xml` file and as such are visible to the author of these changes only. The user-local VM options has the priority over the shared VM options. It means that if anything is written in the field User-local build process VM options, then the field Shared build process VM options is ignored, and the values in the User-local build process VM options field are used instead.

Excludes

File | Settings | Build, Execution, Deployment | Compiler | Excludes for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Excludes for OS X

Use this page to specify files and directories within your project that should not be passed to the compiler.

Item	Keyboard Shortcut	Description
Path		In this field, the path to a file or directory to be excluded from compilation is shown.
Recursively		For a directory: select this option to exclude from compilation all the corresponding subdirectories.
+	Alt+Insert	Use this icon or shortcut to add a file or directory to the list. Select the file or directory in the dialog that opens .
–	Alt+Delete	Use this icon or shortcut to remove the selected item or items from the list.

Note The sources listed on this page, if they are used in the project parts to be compiled (e.g., if they are imported, extended or implemented), will nevertheless be compiled.
– Explicit compiler invocation on excluded directories will force their compilation.

Java Compiler

File | Settings | Build, Execution, Deployment | Compiler | Java Compiler for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Java Compiler for OS X

On the Compiler > Java Compiler page, you can select the Java compiler to be used and specify how it should be used.

- Compiler and bytecode versions
- Javac and Eclipse options
- Ajc options
- Groovy-Eclipse options

Compiler and bytecode versions

Item	Description
Use compiler	Select the compiler to be used: – Javac . The compiler from the JDK that is currently assigned to the project is used. – Eclipse (also known as Eclipse Compiler for Java or ECJ). IntelliJ IDEA comes bundled with the Eclipse compiler. – Groovy-Eclipse . This compiler lets you perform a joint compilation of Groovy and Java codes using the Eclipse compiler. – Ajc (the AspectJ compiler). This option is available only in the Ultimate Edition of IntelliJ IDEA. Besides, the AspectJ compiler is not included in IntelliJ IDEA distribution and should be downloaded separately. See also, Ajc options, Using the AspectJ Compiler (ajc) and Enabling AspectJ Support Plugins .
Project bytecode version	Select the version of bytecode to be generated. (Roughly, this is the minimum target JVM version.) If no particular version is specified, the bytecode version is defined by the compiler. To specify different versions for particular modules, use the controls in the Per-module bytecode version area .
Per-module bytecode version	If necessary, specify the target bytecode versions for individual modules (e.g. if they should differ from that set for the project). Click + and select the modules of interest in the dialog that opens. Then, for each of the modules, click the corresponding Target bytecode version cell and select the version from the list. Use — to remove the selected module or modules from the list.

Javac and Eclipse options

Item	Description
Generate debugging info	If this check box is selected, the compiler generates the information necessary for running the compiled classes in the debugger.
Report use of deprecated features	If this check box is selected, the compiler displays warnings about the deprecated methods, classes, or fields encountered during compilation. (The corresponding warnings are shown in the compiler output window.)
Generate no warnings	If this check box is selected, the compiler omits the warnings about dubious usages of language constructs.
Proceed on errors	For the Eclipse compiler: If you select this check box, the compiler continues the compilation even when compilation errors occur.
Additional command line parameters	Specify the command-line parameters and options to be passed to the compiler at its start. Refer to the compiler documentation for the available options. If you need more room to type, click to open the Additional command line parameters dialog where the text entry area is larger. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m` . – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .

Ajc options

The ajc options are available only in the Ultimate Edition of IntelliJ IDEA.

Item	Description
Path to Ajc compiler	Specify the path to `ajc` (the file `aspectjtools.jar` which is located in `<AspectJ installation directory>\lib`). Type the path in the field, or click ... and select the required file in the dialog that opens .
Test	Click this button to check if the path and the command line parameters are correct. If all is well, the compiler version is displayed. Otherwise, an error message is shown. (Using the path and the parameters specified, IntelliJ IDEA tries to launch the compiler with the additional `-version` parameter.)
Command line parameters	If necessary, specify the command-line options to be passed to the compiler. You can type the parameters right in the field, or click to open the Command line parameters dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` . NOTE: The specified parameters are ignored when `ajc` is used for post-compile weaving.

	See also, Optimizing compilation performance: Using ajc in combination with javac, Controlling the ajc aspectpath and Fine-tuning the use of ajc at a module level.
Generate debug info	If this check box is selected, the compiler generates the information necessary for running the compiled classes in the debugger.
Delegate to Javac	If this option is off, `ajc` is used in all cases. If this option is on, `javac` is used in addition to or instead of `ajc`. For example, `javac` will be used to compile the modules that contain no aspects. As a result, the compilation may become much faster. See also, Optimizing compilation performance: Using ajc in combination with javac and Fine-tuning the use of ajc at a module level.

Groovy-Eclipse options

Item	Description
Path to groovy-eclipse-batch jar	Specify a path to your groovy-eclipse-batch jar location. Click … to open Select Path dialog , select the location of the `.jar` file and click OK. Please note that it is necessary to enter the location of the groovy-eclipse-batch `.jar` file for the compiler to work correctly.
Additional command line parameters	Specify the command-line parameters and options to be passed to the compiler at its start. Refer to the compiler documentation for the available options. If you need more room to type, click to open the Additional command line parameters dialog where the text entry area is larger. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m`. – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`.
Generate debug info	If this check box is selected, the compiler generates the information necessary for running the compiled classes in the debugger.

Annotation Processors

File | Settings | Build, Execution, Deployment | Compiler | Annotation Processors for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Annotation Processors for OS X

Use this page to configure annotation processing for your sources.

Annotation processing profiles

Item	Keyboard Shortcut	Description
+	Keyboard Shortcut	Click this button to create a new profile .
−	Alt+Delete	Click this button to delete the selected profile from the list of existing profiles. All modules, associated with this profile, will be automatically moved to the default profile.
	F6	Click this button to associate a module with a profile . This button only becomes available, when a module in the list of modules under a certain profile gets the focus.

Annotation processors settings

Item	Description
Enable annotation processing	If you want annotation processors to be run during compilation, select this check box and specify associated options: – Obtain processors from project classpath: Select this option, if you want IntelliJ IDEA to obtain the annotation processors from the project classpath. This is useful, for example, if you use a custom annotation processor as part of your project, or if the processor is stored in a `.jar` file attached to all the corresponding modules as a library. – Processor path: Select this option and specify in the field to the right the path to the annotation processor, if it is not desirable to include the processor into the project or project libraries. – Directory name: By default, the sources generated by annotation processors are stored in the project output directory. Using this field, you can override this behaviour for a profile. Specify the name of the directory in which you want IntelliJ IDEA to store the generated sources. The corresponding directories will be created within the content roots of the corresponding modules. At a later time, if you want to use the generated classes as your own sources, you can mark the corresponding directories as source roots. Warning! On rebuild, the directories in which the generated sources are stored will be cleaned up as ordinary output directories. So it is not recommended to store non-generated sources in such directories. Otherwise, the corresponding sources will be lost on rebuild.
Processor FQ Name	Specify the processor fully qualified name. Use + (Alt+Insert) or − (Alt+Delete) to make up the list of annotation processors to be run.
Annotation processor options	If necessary, specify the processor run options either as `-key=value` , or `key=value` . Use spaces to separate individual options. Use + (Alt+Insert) or − (Alt+Delete) to make up the list of options to be passed to the annotation processors.

RMI Compiler

File | Settings | Build, Execution, Deployment | Compiler | RMI Compiler for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | RMI Compiler for OS X

Item	Description
Enable RMI stubs generation	Use this item to analyze compiled classes, searching where the remote interface is implemented. If such classes are not found, IntelliJ IDEA generates stubs for these interfaces incrementally. Tip Only the classes generated by IntelliJ IDEA are analyzed. Classes generated by the other tools are ignored.
Generate IIOP stubs	If enabled, makes the compiler generate IIOP stubs.
Generate debugging info	If enabled, makes the compiler include the information necessary to run this class in the debugger.
Generate no warnings	If this option is enabled, the compiler omits warnings about dubious usages of language constructs.
Additional command line parameters	Enter or edit any additional arguments to be passed to the compiler via the command line. Refer to the compiler documentation for valid options.

Groovy Compiler

File | Settings | Build, Execution, Deployment | Compiler | Groovy Compiler for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Groovy Compiler for OS X

Use this page to configure the Groovy compiler-specific settings.

Item	Description
Path to configscript	Use this field to specify a path to the configuration script parameter to use in Groovy compilation. Type the path in the field, or click ... and select the required file in the dialog that opens .
Invoke dynamic support	Select this check box to activate compilation process that supports " `invokedynamic` " bytecode instruction.
Exclude from stub generation	Use this section to make up a list of files and directories, for which the stub generation step should be omitted. Use + (`Alt+Insert`) to add files and directories to the list, and – (`Alt+Delete`) to delete the selected items.

ActionScript & Flex Compiler

File | Settings | Build, Execution, Deployment | Compiler | ActionScript and Flex Compiler for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | ActionScript and Flex Compiler for OS X

Use the Compiler > ActionScript and Flex Compiler page to select the Flex compiler for your project and to specify the associated settings.

Item	Description
Compile with	Select which of the available Flex compilers should be used: – Built-in compiler shell. An IntelliJ IDEA compiler shell which uses the Flex SDK compiler API. This compiler shell can perform incremental compilations. As a multithreaded shell, it is capable of running a number of compilations simultaneously, in parallel. – Mxmlc/compc. The mxmlc / compc compiler available in Flex SDK. This compiler cannot compile incrementally. However, it can run several independent compilation processes simultaneously which significantly improves the compilation performance. Whichever of the compiler options you use, IntelliJ IDEA keeps track of the modules where nothing has changed since the previous compilation. Consequently, the SWF and SWC files that are up-to-date are not compiled.
Prefer ActionScript Compiler 2.0 for pure ActionScript build configurations	If this check box is selected, the ActionScript Compiler 2.0 (ASC 2.0) is used for pure ActionScript build configurations .
Parallel compilation with up to <this_many> threads or processes	Specify the maximum number of compilation threads (for the built-in compiler shell) or processes (for the mxml/compc compiler) to run simultaneously.
Compiler heap size	Specify the maximum heap size available to the process that launches the compiler. The default 512 Mb is suitable for most of the purposes. However, the built-in compiler shell, as a single-process compiler, may require more memory for large projects.
VM options	If necessary, type the string to be passed to the VM when IntelliJ IDEA launches the compiler. If you need more room to type, click next to the field to access the Flex Compiler VM options dialog where the text entry area is larger. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .

Validation

File | Settings | Build, Execution, Deployment | Compiler | Validation for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Validation for OS X

Use this page to specify the validation settings for your application.

Item	Description
Validate on build	Select this check box, to have IntelliJ IDEA run the desired validators when building your project. The validation results will be shown in the Messages window , and you will be able to easily navigate through the list of issues and jump directly to problematic code fragments.
Validators	Select the validators you want to be run.
Exclude from validation	Use + (Alt+Insert) and — (Alt+Delete) to form the list of files and directories that should be excluded from validation. For directories, use the Recursively check box to specify that the corresponding directory should be excluded from validation along with all the subdirectories contained therein.

Gradle-Android Compiler

File | Settings | Build, Execution, Deployment | Compiler | Gradle-Android Compiler for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Gradle-Android Compiler for OS X

Use this page to specify settings for compiling Android-Gradle projects.

Item	Description
Compile independent modules in parallel (may require larger heap size)	Select this check box if you need to compile independent modules in parallel. For more information, see decoupled projects .
Command-line Options	Use this field to set Gradle command-line options. For more information, see the Gradle command-line options page.
Make project automatically (only works while not running/debugging)	Select this check box to automatically make (compile) the project on every save or autosave.
Use in-process build	Select this check box to use the Gradle in-build process.
Configure on demand	This check box is selected by default. Configuration on demand mode attempts to configure only projects that are relevant for requested tasks. This way, the configuration time of a large multi-project build is greatly improved. For more information, refer to the Gradle configuration on demand page.

Android Compilers

File | Settings | Build, Execution, Deployment | Compiler | Android Compilers for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Android Compilers for OS X

Ctrl+Alt+S

Use this page to configure the behavior of the Android dx tool and the ProGuard tool . This tool converts compiled `.class` files to executable `.dex` files in the Dalvik format for further execution in the Android environment.

On this page:

– DEX

– ProGuard

DEX

In this area, configure the behaviour of the Android dx tool that converts the `.class` files to Dalvik byte code.

Item	Description
Maximum heap size	Use this spin box to control the size of the heap available to the process that launches the compiler. If you are compiling a particularly large or complex project, you may get out-of-memory errors and be required to increase the amount of memory allocated to the compiler.
Additional VM options	In this text box, specify the string to be passed to the Dalvik Virtual Machine for launching the Android application. If necessary, click and type the desired string in the Android DX Compiler VM Options dialog. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Optimize	– Select this check box to have the `.dex` file converted into a `.odex` file (optimized). The optimized `.odex` version of a `.dex` file is stored outside the application package (`.apk`). During the booting of the Android operating system, `.odexa` files make the basis for building a cache for the Dalvik virtual machine. As a result, the operating system in advance knows what applications will be loaded. – When this check box is cleared, the generated `.dex` file is not optimized. By default, the check box is selected.
Force jumbo mode	Select this check box to increase the default number of strings allowed in `.dex` files. By default, the check box is cleared.
Add "--core-library" flag	Select this check box to enable containing classes from certain packages as input files. The packages are as follows: `java` , `javax.accessibility` , `javax.crypto` , `javax.imageio` , `javax.management` , `javax.naming` , `javax.net` , `javax.print` , `javax.rmi` , `javax.security` , `javax.sound` , `javax.sql` , `javax.swing` , `javax.transaction` , `javax.xml` .

ProGuard

In this area, configure the behaviour of the ProGuard tool .

Item	Description
VM options	In this text box, specify the additional options for running the ProGuard tool. For example, to increase the default ProGuard heap size, type `-Xmx<required maximum heap size>` .

Scala Compiler

File | Settings | Build, Execution, Deployment | Compiler | Scala Compiler for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Scala Compiler for OS X

Use this page to configure Scala compiler-specific settings.

Item	Description
Incrementality type	Use this drop-down menu to select the incrementality type for the compiling process. You either can select SBT compiler for your project or IDEA compiler. The incrementality type is set for the whole project excluding profiles.
+	Click this button to create a new profile .
−	Click this button to delete the selected profile from the list of existing profiles. All modules, associated with this profile, will be automatically moved to the default profile.
	Click this button to associate a module with a profile . This button only becomes available, when a module in the list of modules under a certain profile gets the focus.
Compiler order	Use this area to select one of the following compiler options: Note that for more information on what compiler option to choose, see compiling sources for Java/Scala . – Mixed This is a default option. When it is selected, both Java and Scala sources are passed to `Scalac` and then the Java sources are compiled with `Javac` . – Scalac(.*scala, + *java), then Javac(*.java) When you select this button the `Scalac` compiler first compiles `.scala` classes and analyzes `.java` classes. Then the `Javac` compiler is invoked and `.java` classes are compiled. – Javac(*.java), then Scalac(*.scala) When you select this button the `Javac` compiler is invoked before the `Scalac` compiler.
Features	Use this area to configure feature settings .
Options	Use this area to configure options for the Scala compiler.
Compiler plugins	Use this area to add new or delete existing compiler plugins for the `Scalac` compiler. A compiler plugin lets you modify the behavior of the compiler itself without needing to change the main Scala distribution. For more information, see Scala compiler plugins page.

Kotlin Compiler

File | Settings | Build, Execution, Deployment | Compiler | Kotlin Compiler for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Compiler | Kotlin Compiler for OS X

Use this page to configure Kotlin compiler-specific settings.

Item	Description
Generate no warning	If this check box is selected, the compiler won't generate warnings in course of compilation; only errors and info messages will be left.
Additional command line parameters	Specify the command-line parameters and options to be passed to the compiler at its start. Refer to the compiler documentation for the available options. If you need more room to type, click to open the Additional command line parameters dialog where the text entry area is larger. When specifying the parameters and options, follow these rules: – Use spaces to separate individual parameters and options, for example, `-client -ea -Xmx1024m`. – If a parameter or an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"`. – If a parameter or an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"`.
Keep compiler process alive between invocations	If this check box is selected, the compiler process is always alive.
Kotlin to JVM	
Enable precise incremental compilation (experimental)	If this check box is selected, the improved incremental compilation is turned on. Note The incremental compilation is still experimental and may work incorrectly in some cases.
Kotlin to JavaScript	
Generate source maps	If this check box is selected, the compiler generates source maps that set the correspondence between lines in your Kotlin code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.
Output file prefix	Specify the path to the file that will be added as is to the beginning of the generated code. You can enter the path manually, or click and select the required file from the file chooser.
Output file postfix	Specify the path to the file that will be added as is to the end of the generated code. You can enter the path manually, or click and select the required file from the file chooser.
Copy library runtime files	If this check box is selected, the JavaScript files from the libraries will be copied to the folder specified in the field Output directory for library runtime files.
Output directory for library runtime files	This field is only enabled, if the check box Copy library runtime files is selected. Specify here the target folder for the copied files.

Debugger

File | Settings | Build, Execution, Deployment | Debugger for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Debugger for OS X

Use this page to configure behavior of the Debugger and customize its view.

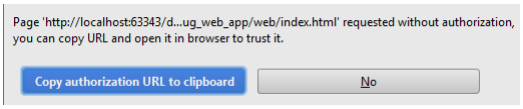
Common options

Item	Description
Focus application on breakpoint	If this check box is selected, on hitting a breakpoint, IntelliJ IDEA will show the location of this breakpoint in the editor and will attempt to bring its frame to the front.
Show debug window on breakpoint	Is this check box is selected, IntelliJ IDEA activates the Debug Tool Window on hitting a breakpoint.
Hide debug window on process termination	Automatically hide the Debug window when the debugged program terminates.
Scroll execution point to center	If this check box is selected, the line with the current execution point will be kept in the middle of the screen.

Java

Item	Description
Transport	Select transport for connection to the process. Available options are socket and shared memory, which is available for Windows systems only.
Force classic VM for JDK 1.3.x and earlier	Check this option to launch process to be debugged with classic VM. Tip For some Java SDK's this check box is disabled, because the `-classic` option should be forced automatically, or when no classic VM is available at all.
Disable JIT	With this option you can control whether the `-Djava.compiler=NONE` parameter is specified when the application is launched. This parameter affects JIT compiler, and if the option is turned on, JIT compiler will be disabled.
Show alternative source switcher	The alternative source switcher appears on top of the editor, if in project there is more than one class with the same fully qualified name.

Built-in server

Item	Description
Port	Use this spin box to specify the number of the port where the built-in web server runs. The port number should be set to 1024 or higher. By default this port is set to the default IntelliJ IDEA port through which IntelliJ IDEA accepts connections from services (`63342`).
Can accept external connections	If this check box is selected, then the files launched on the built-in server running on the specified port, are accessible from a different computer. If this check box is not selected (by default), then only local connections are listened to.
Allow unsigned requests	For security reasons, any request to a page on the built-in server from outside IntelliJ IDEA is by default rejected and the following authorization pop-up window is displayed:  To access the requested page, click Copy authorization URL to clipboard and paste the generated token in the address bar of the browser. However this behaviour may be annoying, for example, it may block your debugging session if manual intervention is impossible. To suppress displaying the authorization pop-up window, select the Allow unsigned requests check box.

Data Views

File | Settings | Build, Execution, Deployment | Debugger | Data Views for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Debugger | Data Views for OS X

Ctrl+Alt+S

Use this page to manage the way data is displayed in the debugger.

On this page:

– Common debugger settings

– Editor

Common debugger settings

Item	Description
Sort values alphabetically	Select this option to sort the values in the Variables pane of the Debug tool window.
Enable auto expressions in Variables view	Select this option if you want the IntelliJ IDEA debugger to automatically evaluate expressions and show the corresponding values in the Variables pane of the Debug tool window. The debugger analyzes the context near the breakpoint (the current statement, one statement before, and one after). It does so to find various expressions in the source code (if available) such as, for example, `myvar.myfield`. If such expressions don't contain explicit method invocations, the debugger evaluates them and shows the corresponding values in the Variables view. **Note** In the languages such as Groovy, it is impossible to tell whether any methods are invoked when evaluating an expression, and such method invocations often cause unwanted side effects, so it is recommended to disable this option when debugging Groovy code.

Editor

Item	Description
Show values inline	Select this option to enable the Inline Debugging feature that allows viewing the values of variables right next to their usage in the editor.

Show value tooltip	Select this option to enable automatic display of tooltips for values.
	A tooltip in this context is a pop-up that provides an alternative, sometimes a more convenient presentation of values in the Variables pane of the Debug tool window.
	
	To illustrate, let's assume that there is a statement like this in your code:
	
	`String s = "Hello, World!\n Hello, World!";`
	
	When this statement is executed in the debugger, you'll see a line looking similar to this in the Variables pane:
	
	`s = {java.lang.String@62} "Hello, World!\n Hello, World!"`
	
	with the line break shown as `\n`.
	
	If the Show value tooltip option is on and you click this line and then hold the mouse pointer on it, you'll see a yellow area (the "tooltip") in which the value of `s` is shown as
	
	`Hello, World!`
	
	`Hello, World!`
	
	with a real line break in place of `\n`.
	
	If this option is disabled, press `Alt` to display a value.

Value tooltips delay (ms)	Specify the delay (in milliseconds) between the moment when the mouse pointer hovers over an object in the Variables pane of the Debug tool window, and the moment when a tooltip with the object's value is displayed.
Show value tooltip on code selection	Select this option to enable tooltips that show the expression value when you select a code fragment in the editor.

Java

Use this page to define the way data is displayed in the Java debugger.

Item	Description
Autoscroll to new local variables	Select this option to automatically scroll to new variables that appear in the scope when stepping.
Show	In this section, select which elements you want the Debugger to display: – Declared type – Synthetic fields – $val fields as local variables – Fully qualified names – Object id – Static fields – Static final fields
Show type for strings	Select this option if you want to show type for pure strings.
Show hex value for primitives	Select this option if you want numeric variables to be displayed in the hexadecimal format.
Hide null array elements	Select this option if you want null array elements to be omitted.
Enable alternative view for Collection classes	Select this option to display collections and maps in a more convenient format.
Enable `toString()` object view	In this section, you can select classes if you need them and their descendants to be presented as a result of the `toString()` method call while debugging. Use the following controls: – For all classes that override toString() method: select this option to show all classes as `toString()`. – For classes from the list: populate the list of classes to be shown as `toString()`, using the , and the buttons. Use the check boxes next to the class names to temporarily enable or disable particular filters. – : click this button to add a class to the list using the Choose Class dialog. – : click this button to add a custom class filter using the New Filter dialog. To define a filter, enter a string pattern, e.g. `*.Test`, `javax.swing.*`, etc. – : click this button to remove a filter from the list.

Java Data Type Renderers

IntelliJ IDEA allows you to specify how different objects are displayed in the debugger on a class-by-class basis. You can assign the expressions to be displayed rather than rely on the object's String representation.

For example, if an object represents a user, you may want to see users represented by their login name; or, for a cache entry object, its age and contents may be appropriate. IntelliJ IDEA refers to these as type renderers.

If no rendering scheme is defined, this dialog does not show any controls. To start working with renderers click +.

Item	Description
+	Use this icon to add a new rendering scheme to the list.
−	Use this icon to remove the selected scheme from the list.
	Use this icon to create a copy of the selected scheme.
	Use these icons to move the selected item one line up or down in the list. Note that the order of items in the list determines which renderer is used in the case of ambiguity born of class inheritance.
Renderer name	In this field, specify the name of a new renderer, or edit an existing renderer name.
Apply renderer to objects of type (fully-qualified name)	In this field, specify the type of objects that will be represented with this renderer. Type a fully qualified object name, or click the Browse button ... and choose the desired type from the list in the Renderer Reference Type dialog.
When rendering a node	This option determines how an object is displayed in the debugger when nodes are collapsed: – Use default renderer: select this option to display the node in the default way. – Use following expression: enter the java expression you want to use to identify an object. You can use object properties, constants, and even a string math as part of your renderer. Note that you can use code completion (Ctrl+Space) when defining expressions. All method calls and member variable access are relative to the object you're rendering. Use `this` to refer to an instance to which the renderer applies. Warning! – Using heavy expressions in renderers may slow down data rendering in views. – Method calls should be used with caution because of possible side-effects.
When expanding a node	This option determines how an object is displayed in the debugger when nodes are expanded: Normally, expanding a node in the debugger lists the object's member variables (using the renderer appropriate for the corresponding object types). This option lets you override this behavior and select a single expression or a series of expressions to be displayed. You may use this to limit the amount of information displayed, or to be more precise in how the information is presented. – Use default renderer: select this option to display the node children in the default way. – Use following expression: enter the Java expression you want to use to identify an object. Test if a node can be expanded (optional): enter a Boolean expression. If it is `true`, the renderer displays expandable nodes for the defined objects. Otherwise, no nodes are displayed. – Use list of expressions: create a list of separate expressions to be calculated and presented as node children. Use: + (Alt+Insert) to create a new expression. − (Alt+Delete) to remove the selected expression from the list. ↑ (Alt+Up) to move the selected expression one line up in the list. ↓ (Alt+Down) to move the selected expression one line down in the list. Note that you can use code completion (Ctrl+Space) when defining expressions.

JavaScript

File | Settings | Build, Execution, Deployment | Debugger | Data Views | JavaScript for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Debugger | Data Views | JavaScript for OS X

`Ctrl+Alt+S`

Use this page to configure JavaScript debug options.

ItemDescription	
Show the following properties for an object node	Select this check box if you want IntelliJ IDEA to show certain object node properties and configure a list of the properties to display. For each object node in the Variables pane, IntelliJ IDEA will display a label with the values of the listed properties.

ItemShortcutDescription		
+	`Alt+Insert`	Click this button to add a new property.
−	`Alt+Delete`	Click this button re remove the selected property.

Kotlin

File | Settings | Build, Execution, Deployment | Debugger | Data Views | Kotlin

Use this page to turn on or off showing the values of the delegated properties.

Item	Description
Calculate values of delegated properties (may affect program execution)	If this check box is selected, the values of the delegated properties are shown in the list of an object's properties in the debugger.

Stepping

File | Settings | Build, Execution, Deployment | Debugger | Stepping for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Debugger | Stepping for OS X

Ctrl+Alt+S

Use this page to improve the debug stepping speed and specify the elements to be skipped while stepping.

Item	Description
Groovy	
Do not step into specific Groovy classes	Select this check box if you don't want to step into the `org.codehaus.groovy.*` and `groovy.*` Groovy classes while debugging.
Java	
Skip synthetic methods	Select this check box to suppress stepping into synthetic methods (methods generated by the compiler) while debugging.
Skip constructors	Select this check box to suppress stepping into constructors while debugging.
Skip class loaders	Select this check box to suppress stepping into class loaders while debugging.
Skip simple getters	Select this check box to suppress stepping into simple getter methods (i.e., methods designed just to return the necessary value) while debugging.
Do not step into the classes	Select this check box to suppress stepping into the specified classes while debugging. The list of classes contains entries of two types: – Fully qualified class names. – Patterns with wildcard characters (*). By default, the list contains some standard Java SDK class patterns so that you do not have to waste your time stepping into Java class libraries. Use the check boxes in the list to disable/enable particular patterns temporarily. Use the , , and buttons to manage the list.
Evaluate finally blocks on pop frame	Select whether you want to evaluate `finally` blocks on pop frame or not, or you want to be notified before they are evaluated.
Resume only the current thread	Select this check box, if you need to resume only the active thread when stepping.
JavaScript	
Do not step into library scripts	Select this check box to suppress stepping into library scripts while debugging.
Do not step into scripts	Select this check box to suppress stepping into certain scripts while debugging. Use the toolbar buttons to manage the list of scripts to be skipped.
Kotlin	
Do not step into specific Kotlin classes	Select this check box if you don't want to step into specific Kotlin classes while debugging.

Item	Shortcut	Description
+	Alt+Insert	Click this button to add a new script filter.
–	Alt+Delete	Click this button to delete the selected filter from the list.
	Enter	Click this button to edit the selected filter.
/		Use these buttons to arrange filters as required.
		Click this button to create a copy of the selected filter.

Live Edit

File | Settings | Build, Execution, Deployment | Debugger | - Live Edit for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Debugger | - Live Edit for OS X

Ctrl+Alt+S

The page is available only when the LiveEdit plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Use this page to enable and disable instant preview of HTML, CSS, and JavaScript files in the actual browser and configure the appearance of the preview.

Item	Description
Highlight current element in browser on caret change	When this check box is selected, the browser highlights the area that represents the code you are currently editing in IntelliJ IDEA.
Update	In this area, configure the way changes made to the code during a debugging session are applied. Note that an update will now be performed only if none of the modified files have any syntax errors. – Auto in (ms): Choose this option to have the changes applied automatically at certain time interval and specify the delay in ms in the text box. This policy is not available for the client-side code of Meteor applications. – Manual: Choose this option to apply the changes manually by clicking the Update <run configuration name> JavaScript button or the Update <run configuration name> button on the toolbar of the Debug tool window. It is recommended that you choose this option for debugging Meteor applications because applying changes to the client-side code is not supported. – Restart if hotswap fails: With changes in HTML, CSS, and JavaScript on the client side, the contents of a Web page in the browser are updated without reloading. For NodeJS or Meteor applications, IntelliJ IDEA first tries to update the application incorporating the changes without restarting the NodeJS server. – Select this check box to have IntelliJ IDEA try restarting the server if the changes cannot be applied automatically. After the restart, IntelliJ IDEA brings you to the execution point where the problem took place. If even with this option chosen automatic upload still fails, tou will have to restart the server manually by clicking the Rerun <run configuration name> button . – When the check box is cleared, IntelliJ IDEA just displays a pop-up window informing you about the failure and suggesting you to restart the server manually.

HotSwap

File | Settings | Build, Execution, Deployment | Debugger | HotSwap

Use this tab to manage the behavior of the HotSwap mechanism.

Item	Description

Java

In this section, specify the behavior of HotSwap for Java:

Item	Description
Make project before reloading classes	This option controls Run | Reload Changed Classes action behavior. If it's turned on, the make process is performed before searching and reloading changed classes.
Enable "JVM will hang" warning	If this check box is selected, then, on an attempt to perform HotSwap while JVM is suspended, a warning about the possible hanging of this JVM will be displayed.
Reload classes in background	Select here whether you want to reload classes in background mode. This means, all progress messages will be displayed in a status bar.
Reload classes after compilation	Use the controls in this area to configure behavior of the HotSwap mechanism. – Always - select this option to have classes reloaded automatically. – Never - when this option is selected, classes are not reloaded at all because the HotSwap mechanism is inactive. – Ask - select this option to have IntelliJ IDEA ask you whether to reload altered classes or not.

Groovy

In this section, specify the behavior of HotSwap for Groovy:

Item	Description
Enable hot-swap agent for Groovy code	If this check box is selected, then a special agent will be added to the debugged process in order to enable hot-swap for Groovy code. If you don't want hot-swap, or this agent is getting in the way, clear this check box. Note Note that enabling hot-swap agent may cause serialization problems in the debugged applications.

Deployment

File | Settings | Build, Execution, Deployment | Deployment for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Deployment for OS X

`Ctrl+Alt+S`

In this section:

- Deployment
  - Basics
  - Toolbar and common options
- Deployment: Connection Tab
- Deployment: Mappings Tab
- Deployment: Excluded Paths Tab
- Add Server Dialog
- Options
- Advanced Options Dialog
- Files/Folders Default Permissions Dialog

Basics

On this page, create, edit, and delete server access configurations that give you control over interaction between IntelliJ IDEA and servers. Anytime you are going to use a server, you need to define a server access configurations, no matter whether your server is on a remote host or on your computer.

Among numerous ways to configure your development and production environments the most frequent ones are as follows:

- The Web server is installed on your computer. The sources are under the server document root (for example, `/htdocs`), and you do your development right on the server.
- The Web server is installed on your computer but the sources are stored in another folder. You do your development, then copy the sources to the server.
- The Web server is on another computer (remote host). Files on the server are available through the FTP/SFTP/FTPS protocol, through a network share, or a mounted drive.

Note that IntelliJ IDEA assumes that all development, debugging, and testing is done on your computer and then the code is deployed to a production environment. For detailed reasoning of this approach, see Deployment. Working with Web Servers

Let's define the terms and their meaning in the context of synchronization between IntelliJ IDEA and servers.

- An in-place server is a server whose document root is the parent of the project root, either immediate or not. In other words, the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.
- A local server is a server that is running in a local or a mounted folder and whose document root is NOT the parent of the project root.
- A remote server is a server on another computer (remote host).
- The server configuration root is the highest folder in the file tree on the local or remote server accessible through the server configuration. For in-place servers, it is the project root.
- A local file/folder is any file or folder under the project root.
- A remote file/folder is any file or folder on the server, either local or remote.
  Suppose you have a project `C:/Projects/My_Project/` with a folder `C:/Projects/My_Project/My_Folder` and a local server with the document root in `C:/xampp/htdocs`. You upload the entire project tree to `C:/xampp/htdocs/My_Project`. In the terms of IntelliJ IDEA, the folder `C:/Projects/My_Project/My_Folder` is referred to as local and the folder `C:/xampp/htdocs/My_Project/My_Folder` is referred to as remote.
- Upload is copying data from the project TO the server, either local or remote.
- Download is copying data FROM the server to the project.

Synchronization with servers, uploading, downloading, and managing files on them are provided via the Remote Hosts Access bundled plugin, which is by default enabled. If not, activate it in the Plugins page of the Settings dialog box. For details, see Enabling and Disabling Plugins .

Toolbar and common options

Use the toolbar buttons to manage the list of configurations.

The left-hand pane shows a list of all the server access configurations available in IntelliJ IDEA. When you select a configuration, the right-hand pane shows the configuration details.

Item	Tooltip and shortcut	Description
+	Add `Insert`	Click this button to open the Add Server dialog box and define a new configuration there.
–	Delete `Delete`	Click this button to remove the selected configuration from the list.
	Copy `Ctrl+D`	Click this button to copy the settings of the selected configuration.
	Use as Default	Click this button to have IntelliJ IDEA apply the settings of the selected configuration by default during automatic upload of changed files.

Deployment: Connection Tab

Use this tab to choose the way to access the Web server and specify the connection settings .

Item	Description
Name	The text box shows the configuration name specified in the Add Server dialog box. Edit the configuration name, if necessary.
Visible only for this project	Use this check box to configure the visibility of the server access configuration (deployment configuration). – Select the check box to restrict the use of the configuration to the current project. Such configurations cannot be reused outside the current project, they do not appear in the list of available configurations in other projects. For example, if this check box is selected in an SFTP configuration, you cannot use your SSH credentials from it when you configure a remote interpreter. – When the check box is cleared, the configuration is visible in all IntelliJ IDEA projects and the settings from, including SSH credentials, can be reused. See Configuring Node.js Interpreters and Configuring Remote PHP Interpreters for details.
Access type	From this drop-down list, choose the way to access the server. Use the Up and Down keyboard keys to scroll through the list of server configuration types. The available options are: – FTP: choose this option to have IntelliJ IDEA access the server via the FTP file transfer protocol . – SFTP: choose this option to have IntelliJ IDEA access the server via the SFTP file transfer protocol. – FTPS: choose this option to have IntelliJ IDEA access the server via the FTP file transfer protocol over SSL (the FTPS extension). – Local or mounted folder: choose this option if the Web server is running in a local or a mounted folder and its document root is NOT the parent of the project root. – In-place: choose this option if the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.

Upload/Download Project Files

In this area, specify the settings for accessing the server to upload and download files to and from.

The set of controls in the area depends on the chosen server access type.

Item	Description	Available for
Folder	In this field, specify the server configuration root. The server configuration root is the highest folder in the file tree on the server that can be accessed through the server configuration. The easiest way is to use the document root of your Web server as defined in the Web server configuration file. However you can appoint any other existing folder under the document root.	
FTP/FTPS/SFTP host	In this text box, specify the host name of the FTP/SFTP server to upload the files to.	
Port	In this text box, specify the port to use. The default values are: – 21 for FTP and FTPS – 22 for SFTP	
Root Path	In this text box, specify the server configuration root relative to your user home which was defined when you registered your account. This folder will be the highest one in the folder structure accessible through the current server configuration. Do one of the following: – Accept the default value `/`, which points at the user home folder on the server. – Type the path manually. – Click the Browse button ... and select the desired folder in the Choose Root Path dialog box that opens. – Click the Autodetect button and have IntelliJ IDEA detect the user home folder settings on the FTP/SFTP server and set up the root path according to them. The button is only enabled when you have specified your user name and password.	
Autodetect	Click this button to have IntelliJ IDEA detect the user home folder settings on the FTP/SFTP server and set up the root path according to them.	
User name	In this text box, type your user name for authentication to the server. Note The button is only enabled when you have specified your user credentials.	
Log in as anonymous	Select this check box to enable anonymous access to the server with your email address as password.	
Auth type	From this drop-down list, select the client authentication method. The available options are: – Password - select this option to use standard authentication through a password. – Key pair (OpenSSH) - select this option to use SSH authentication via a key pair. To apply this authentication method, you need to have your private key on the client machine and your public key on the remote server you connect to. IntelliJ IDEA supports private keys generated using the OpenSSH utility. See http://wiki.qnap.com/wiki/How_To_Set_Up_Authorized_Keys for details.	
Password	In this text box, type your password for authentication to the server.	
Private key file	In this text box, specify the location of your private key file.	
Passphrase	In this text box, specify your authentication passphrase.	
Save password	Select this check box to have IntelliJ IDEA remember the specified password.	
Save passphrase	Select this check box to have IntelliJ IDEA remember the specified passphrase.	
Test FTP/FTPS/SFTP connection	Click this button to check that the specified settings ensure successful connection via FTP/SFTP.	
Explicit	Choose this option to have the explicit (active) security applied. Immediately after establishing connection, the FTP client on your machine sends a command to the server to establish secure control	

	connection through the default FTP port.
Implicit	Choose this option to have the implicit (passive) security applied. In this case, security is provided automatically upon establishing connection to the server which appoints a separate port for secure connections.
Advanced options	Click this button to specify additional uploading settings in the Advanced Options dialog box that opens.
Web server root URL	In this text box, specify the URL address of the Web server root folder. Both the HTTP and the HTTPS protocols are supported. To access a server through HTTPS, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized authority and import this certificate in the truststore/keystore of the Oracle JRE (Java Runtime Environment) on which IntelliJ IDEA runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

```
<jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_name>.cert>
```

   If you are using the Oracle JRE bundled with IntelliJ IDEA, the default path to the truststore/keystore is `<%product_installation_folder>/jre/jre/lib/security/jssecacerts` or `<%product_installation_folder>/jre/jre/lib/security/cacerts`.
   Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts`.

2. When asked to enter a password for the truststore/keystore, specify the default one `changeit`.

3. Open the `IntelliJ IDEA.exe.vmoptions` file in the `<IntelliJ IDEA_installation_folder>/bin` and add the following line to it:

```
-Djavax.net.ssl.keyStore=<path to keystore>
```

4. Restart IntelliJ IDEA.

Learn more at Java6 and Java7.

Open	Click this button to make sure that the specified server root URL address is accessible and points at the correct Web page.

Deployment: Mappings Tab

In this tab, configure mappings, that is, set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . The easiest way is to map the entire project root folder to a folder on the server, whereupon the project folder structure will be repeated on the server, provided that you have selected the Create Empty directories check box in the Options dialog box . "For more details, see Customizing Upload/Download . Below are definitions of terms used in this topic in the context of of synchronization between IntelliJ IDEA and servers.

- A local server is a server that is running in a local or a mounted folder and whose document root is NOT the parent of the project root.
- A remote server is a server on another computer (remote host).
- The server configuration root is the highest folder in the file tree on the local or remote server accessible through the server configuration. For in-place servers, it is the project root.
- A local file/folder is any file or folder under the project root.
- A remote file/folder is any file or folder on the server, either local or remote.

  Suppose you have a project `C:/Projects/My_Project/` with a folder `C:/Projects/My_Project/My_Folder` and a local server with the document root in `C:/xampp/htdocs` . You upload the entire project tree to `C:/xampp/htdocs/My_Project` . In the terms of IntelliJ IDEA, the folder `C:/Projects/My_Project/My_Folder` is referred to as local and the folder `C:/xampp/htdocs/My_Project/My_Folder` is referred to as remote.

Item	Description
Use this server as default	Click this button to have IntelliJ IDEA apply the settings of the selected configuration by default during automatic upload of changed files. This button is only enabled for the non-default servers; for the server used as default , this button is disabled.
Local Path	In this text box, specify the full path to the desired folder in the project tree. In the simplest case it is the project root. Type the path manually or click the Browse button and select the desired location in the Choose Local Path dialog box that opens.
Deployment Path	In this text box, specify the folder on the server where IntelliJ IDEA will upload the data from the folder specified in the Local Path text box. Type the path to the folder relative to the server configuration root. If the folder with the specified name does not exist yet, IntelliJ IDEA will create it, provided that you have selected the Create Empty directories check box in the Options dialog box . For more details, see Customizing Upload/Download . The text box is not available for server configurations of the type In-place.
Web Path	In this text box, type the path to the folder on the server relative to the server configuration root. Actually, type the relative path you typed in the Deployment Path text box.
Add	Click this button to have a new line added to the list of mappings.
Remove	Click this button to remove the selected mapping from the list.

Deployment: Excluded Paths Tab

Use this tab to configure a list of local and remote folders that you do not want to be involved in upload/download.

Item	Description
Add local path	Click this button to have an empty line added to the list and specify the location of the folder to be protected against upload/download. Type the path manually or click the Browse button ... and choose the required folder in the dialog that opens .
Add deployment path	Click this button to have an empty line added to the list. Click the Browse button ... The Select remote excluded path dialog box that opens shows the data on the host accessed through the selected server configuration. Select the required folder.
Remove path	Click this button to remove the selected item from the list. The button is only available when a line is selected.

Add Server Dialog

The dialog box opens when you click the Add toolbar button + on the Deployment page.

Use the dialog box to create server access configurations that give you control over interaction between IntelliJ IDEA and servers. Anytime you are going to use a server, you need to define a server access configurations, no matter whether your server is on a remote host or on your computer.

Item	Description
Name	In this text box, type the name of the new Web server configuration.
Type	From this drop-down list, choose the way to access the server. Use the Up and Down keyboard keys to scroll through the list of server configuration types. The available options are: – FTP: choose this option to have IntelliJ IDEA access the server via the FTP file transfer protocol . – SFTP: choose this option to have IntelliJ IDEA access the server via the SFTP file transfer protocol. – FTPS: choose this option to have IntelliJ IDEA access the server via the FTP file transfer protocol over SSL (the FTPS extension). – Local or mounted folder: choose this option if the Web server is running in a local or a mounted folder and its document root is NOT the parent of the project root. – In-place: choose this option if the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.

When editing the server configuration name in the Name text box, use the Up and Down keys on your keyboard to change the preselected server access to type in the Type drop-down list.

Options

File | Settings | Build, Execution, Deployment | Deployment | Options for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Deployment | Options for OS X

`Ctrl+Alt+S`

Tools | Deployment | Options

Use this page to specify additional configuration settings for uploading and downloading project files to and from local and remote servers. For more details about various server access configurations, see Deployment. Working with Web Servers .

The options specified in this dialog box apply to all defined server configurations regardless of the server type (local, remote) and the data transfer protocol used. Protocol-specific options for server configurations of the type FTP/SFTP/FTPS are defined in the Advanced Options Dialog .

Item	Description
Exclude items by name	In this text box, specify patterns for the names of files and folders that you do not need to be deployed. Use semicolons as delimiters. Wildcards are welcome. The exclusion is applied recursively. This means that if a matching folder has subfolders, the contents of these subfolders are not deployed either.
Operations logging	Use this drop-down list to specify how much detailed logging you need to have. The available options are: – Errors only: select this option to have the log show only errors occurred during upload. – Brief: select this option to have all events reflected in the log but without details. – Detailed: select this option to have more details on the upload shown in the log, for example, full file paths.
Stop operation on the first error	Select this check box to have data transfer stopped as soon as an error occurs.
Overwrite up-to-date files	Do one of the following: – Select this check box to have all the files uploaded no matter whether they have been changed since the previous upload or not. – Clear the check box to upload only files that have been changed since the previous upload.
Preserve files timestamps	Select this check box to prevent resetting timestamps of files on upload.
Delete target items when source ones do not exist	If this check box is selected, any file in the destination directory will be removed if the file with this name is not involved in the current upload. This option is applicable when synchronization if performed from the Project tool window or from the Remote Host tool window.
Create empty directories	Select this check box to have an empty directory on the server created automatically if a new local directory has been created in your project since the last upload in the source folder.
Prompt when overwriting or deleting local items	Select this check box to have IntelliJ IDEA ask you for confirmation before overwriting or deleting local items for synchronization during download.
Upload changed files automatically to the default server	From this drop-down list, choose when you want IntelliJ IDEA to automatically upload a file to the default server. The available options are: – Always: choose this option to have a file uploaded upon each save, no matter automatic or explicitly invoked. – On explicit save action: choose this option to have a file uploaded after save only if this save was invoked manually by choosing File | Save all or pressing `Ctrl+S` . – Never: choose this option to suppress automatic upload. The default server configuration is appointed on the Deployment page by selecting the desired configuration in the list and clicking the Use as Default toolbar button .
Upload external changes	Select this check box to have IntelliJ IDEA upload also the local changes that were made using a third-party tool.
Override default permissions on files	Select this check box to change the default permissions assigned to uploaded files on remote hosts. Click the Browse button ... to open the Files Default Permissions dialog box, where you can manage access to uploaded files on remote hosts by assigning permissions.
Override default permissions on folders	Select this check box to change the default permissions assigned to uploaded folders on remote hosts. Click the Browse button ... to open the Folders Default Permissions dialog box, where you can manage access to uploaded folders on remote hosts by assigning permissions.
Warn when uploading over newer file	Use this drop-down list to define the version-control policy to apply when uploading files to remote hosts. Depending on this choice, IntelliJ IDEA either checks whether any changes have been made to the corresponding files on the remote host since you downloaded them or just overwrites the remote files. – No: choose this option to have the file on the remote host overwritten with its local copy silently. All the changes made to the remote file since your last synchronization will be abandoned. – Compare timestamp and size: if you choose this option, IntelliJ IDEA performs two checks: 1. Compares the sizes of the local and remote files. 2. Compares the remote file timestamp set at the moment of the last synchronization with the current remote file timestamp. If the files differ in their size or the remote file timestamps differ, IntelliJ IDEA opens a Difference Viewer for Files , where you can explore and integrate the differences. This type of check depends on the timezone setting. If the timezone setting on your local machine is different from that on the remote host, the check may be successful even though the file versions actually differ. – Compare content: when this option is chosen, IntelliJ IDEA compares the content of the local and remote files. If any diversions are detected, IntelliJ IDEA opens a Difference Viewer for Files , where you can explore and integrate the differences.
Notify about remote changes	Select this check box to receive notifications about changes on the remote host. The check box is available only when the Compare timestamp and size: or Compare content: option is selected in the Warn when uploading over newer file drop-down list.
SFTP advanced options (IDE level)	
Add new host key to known_hosts	Choose whether IntelliJ IDEA should ask about connecting to a host not mentioned in the file `known_hosts` . The following options are available: – Always: always connect and add its record to the file `known_hosts` . – Ask: ask whether IntelliJ IDEA should connect to a host and add its record to the file `known_hosts` – Never: do not connect.
Hash hosts in	If this check box is selected, the new host record will be stored in hash format.

known_hosts file

Advanced Options Dialog

The dialog box opens when you click the Advanced Options button in the Connection tab of the Deployment page. Use this dialog box to customize upload/download by specifying additional protocol-specific options for server configurations of the type FTP/SFTP/FTPS.

Item	Description	Available for
Passive mode	Select this check box to set the client on your machine to the passive mode , when it connects to the server to inform about being in the passive mode, receives the port number to listen to, and established data connection through the port with the received number. This mode is helpful when your machine is behind a firewall.	FTP, FTPS
Show and process hidden files	When this check box is selected: 1. Hidden files and directories are shown in the Remote Host Tool Window . 2. Hidden files and directories are involved in diff and synchronization operations. The name of a hidden file or directory starts with a dot (`.`).	FTP, FTPS
Compatibility mode	Select this check box to ensure compatibility in child file naming with your FTP server. – This option is helpful if the remote FTP server reports the following error: `Invalid descendant file name <file name>` – Selecting this option may slow down synchronization with the server.	FTP
Retrieve accurate files timestamps	Use this drop-down list to specify the MDTM FTP command calling policy to retrieve the last-modified time of a given file on the remote host. The available options are: – Always - select this option to have MDTM called for every file shown in the Remote Host tool window. – On copy - select this option to have MDTM called in the following cases: – To check whether a file is up to date when the Overwrite up-to-date files check box in the Options dialog box is cleared. – To preserve the actual time stamp od a file during download. – Never - select this option to suppress calling MDTM.	FTP, FTPS, SFTP
Limit concurrent connections	Select this check box to have IntelliJ IDEA restrict the number of connections to be supported simultaneously and specify the maximum number of allowed connections in the text box.	FTP, FTPS, SFTP
Control encoding	In this text box, specify the encoding that matches the encoding used by your server. Accept the default value if you are not sure that it supports UTF-8 encoding.	FTP, FTPS, SFTP
Always use LIST command	Select this check box to use the standard `LIST` command for listing instead of the `MLSD` command. This lets you avoid problems, for example, failure during upload with the Invalid descendent file name exception if the FTP server supports `MLSD` and returns `cdir` .	FTP, FTPS
Send keep alive message each	In this text box, specify how often you want IntelliJ IDEA to send commands to the server to reset the timeout and thus preserve the connection.	FTP, FTPS, SFTP
Use keep alive command	From this drop-down list, choose the commands to be sent to the server to reset the timeout and thus preserve the connection.	FTP, FTPS

Files/Folders Default Permissions Dialog

This dialog opens when you select the Override default permissions on files or Override default permissions on folders check box in the Options dialog and click the Browse button next to it.

Use this dialog box to re-assign default server permissions to owners of files or folders, groups of owners, and other users.

The following identifiers are used for permission types:

- R stands for Read.
- W stands for Write.
- X stands for Execute

Item	Description
Owner	In this row, specify what an owner of a file or folder may do by selecting the check boxes below the corresponding identifiers.
Group	In this row, specify what a group of owners of a file or folder may do by selecting the check boxes below the corresponding identifiers.
Others	In this row, specify what any one else may do by selecting the check boxes below the corresponding identifiers.
Octal	In this text box, specify the octal representation of the specified set of permissions. By default, IntelliJ IDEA calculates and re-calculates the value of the field as you select or clear the desired check boxes. You can also specify the octal value manually.

Arquillian Containers

Specify the containers for running your Arquillian tests.

Toolbar

Icon	Description
+	Add a container configuration. Select one of the following: – Manual container configuration. Select this option to configure a managed or remote container. You can also use this option to configure an embedded container if the container of interest is not present in the Embedded list. Use +, —, and in the Dependencies section to specify the container implementation. – Embedded. Select this option to use an embedded container. Then select the container of interest. Use in the Dependencies section to fine-tune the container configuration.
—	Remove the selected container configurations.

Container settings

Item	Description
Name	The name of the container configuration.
Arquillian container	For an embedded container: a link to a web page for the corresponding container adapter Maven artifact.
Dependencies	The container adapter implementation represented by a Maven artifact or IntelliJ IDEA library . + Add a Maven artifact or IntelliJ IDEA library. — Remove the selected items from the list. Edit the settings for the selected item. Depending on the container, you can change the adapter version or choose to download the container source code and documentation.
VM options	Options and arguments to be passed to the JVM in which the container runs. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Environment variables	The environment variables to be passed to the corresponding JVM.

Application Servers

File | Settings | Build, Execution, Deployment | Application Servers for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Application Servers for OS X

From the Run/Debug Configurations dialog: Configure when editing a run/debug configuration for an application server.

Use this page or dialog to manage configurations for the supported application servers and their settings.

The servers that are already defined in IntelliJ IDEA are shown in the pane under + and —. When you select a server in this pane, the corresponding configuration settings and associated controls are shown in the area to the right. (If you are editing a run configuration for a particular server (e.g. Tomcat), only the existing configurations for this particular server are shown.)

Use + to create server configurations and — to remove them.

By the time you start creating a server configuration, the corresponding server must already be installed on your computer. (This isn't necessary for CloudBees .)

Note that the list of servers you can work with depends on which server integration plugins are currently enabled. See Enabling Application Server Integration Plugins .

– Toolbar
– Main settings and controls
– Libraries
– Additional Libraries for Frameworks
– Configuration file list (for Jetty 7 or later versions)

Toolbar

Item	Description
+ (`Alt+Insert`)	Use this icon or shortcut to create a server configuration (i.e. to define a server in IntelliJ IDEA): 1. If the server list is shown, select the server of interest. 2. In the dialog that opens, specify the server settings and click OK. For most of the servers, you just need to specify the server installation directory (referred to as the server home). For more information, see Main settings and controls.
— (`Alt+Delete`)	Use this icon or shortcut to remove the selected application server configuration.

Main settings and controls

Item	Description
Name	The server configuration name.
<Server> Home	The path to the application server installation folder. Click … and select the folder in the dialog that opens .
<Server> Version	The detected application server version (readonly).
<Server> base directory (for Jetty, Tomcat and TomEE)	The path to the server base directory. Click … and select the directory in the dialog that opens .
Register schemas (for JBoss)	To be able to validate JBoss XML configuration files (such as `standalone.xml` , `domain.xml` , etc.) when editing them in IntelliJ IDEA, you can register XML schemas (XSDs) available in the JBoss installation: 1. Click Register schemas. 2. To register all the schemas, just click OK in the dialog that opens. If there are schemas that you don't want to register, select those schemas under External Schemas and DTDs, click — and then click OK. To view or edit the schema list at a later time, use the Schemas and DTDs page in the Settings dialog .
Download client libraries (for CloudBees)	The Tomcat instance embedded in CloudBees is used as a server. This Tomcat instance is included in the CloudBees client libraries which you can download. If the button is inactive, the client libraries (along with the corresponding Tomcat instance) are already available in IntelliJ IDEA.

See also, Libraries, Additional Libraries for Frameworks and Configuration file list (for Jetty 7 or later versions).

Libraries

When you create a server configuration, normally, an associated application server library is created. As a rule, this library includes Servlet, JSP and EJB implementations available in the server distribution.

Use the controls under Libraries to manage the contents of the library.

Item	Description
+ (`Alt+Insert`)	Use this icon or shortcut to add items (classes, sources, documentation, etc.) to the library. In the dialog that opens , select the necessary files and folders. These may be individual `.class` and `.java` files as well as directories and archives (`.jar` and `.zip`) containing such files. IntelliJ IDEA will analyze the selected files and folders, and automatically assign their contents to the appropriate library categories (Classes, Sources, Documentation, etc.). When IntelliJ IDEA cannot guess the category (e.g. when you select an empty folder), a dialog will be shown, in which you will be able to specify the category yourself.

	To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.
	Click this icon to make certain library items "excluded" (see Excluded library items). In the dialog that opens, select the items that you want IntelliJ IDEA to ignore (folders, archives and folders within the archives), and click OK.
– (Delete)	When you click this icon or press Delete : – The selected "ordinary" library items are removed from the library. – The selected excluded items (see Excluded library items) become "ordinary" items, i.e. their excluded status is cancelled. The items themselves will stay in the library.

Additional Libraries for Frameworks

For certain application server configurations (e.g. GlassFish), there is an Additional Libraries for Frameworks list with check boxes.

Each item in this list, potentially, is a library that implements the corresponding framework. (Corresponding library files are included in the server distribution.)

When you select a check box in the list, the corresponding application server library is created. The dialog that opens lets you select the modules in which this library should be used. (As a result, the library is added to dependencies of the selected modules.)

Configuration file list (for Jetty 7 or later versions)

For Jetty 7 or later versions, there is a section where the server configuration (.xml) or module (.mod) files are listed.

This section provides a GUI for editing the file `<jetty_home>\start.ini` . (`start.ini` contains the options for `<jetty_home>\start.jar` which is used to start Jetty.)

Use the check boxes to make the files active or inactive. Use +, –, ✎, ↑ and ↓ to add, remove, replace and reorder the files.

Item	Description
Active	Use the check boxes to make the files active or inactive. As soon as you deselect a file, the corresponding line in `start.ini` is commented. When you make a file active, the corresponding entry in `start.ini` is uncommented.
Path	The paths to the corresponding files are shown (readonly). For the files within the Jetty installation directory, the paths are relative to the installation directory. For all the rest of the files, the absolute paths are shown.
+ (Alt+Insert)	Use this icon or shortcut to add a file to the list. In the dialog that opens , select the necessary file and click OK. Adding a file to the list leads to adding a new (commented) entry to `start.ini` .
– (Alt+Delete)	Use this icon or shortcut to remove the selected file from the list. Note that this operation does not delete the file physically. However, the corresponding entry is removed from `start.ini` .
✎ (Enter)	Use this icon or shortcut to replace the selected file. In the dialog that opens , select the replacement file and click OK.
↑ (Alt+Up)	Use this icon or shortcut to move the selected file one line up in the list.
↓ (Alt+Down)	Use this icon or shortcut to move the selected file one line down in the list.

Clouds

File | Settings | Build, Execution, Deployment | Clouds for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Clouds for OS X

Use this page to manage your cloud access configurations. Each configuration includes your cloud user account info and related settings.

To create a new configuration, click + and select the cloud platform of interest. (By the time you start creating a cloud access configuration, you must already have the corresponding cloud user account. See Working with Cloud Platforms .)

If the Docker integration plugin is installed, this page also lets you manage your Docker configurations.

- CloudBees
- Cloud Foundry
- Google App Engine
- Heroku
- OpenShift
- Docker
- Toolbar

CloudBees

Specify your CloudBees user account details and related settings. For additional information, see the CloudBees documentation .

Item	Description
Name	The name of the CloudBees configuration.
Domain	Your CloudBees domain.
Username (email)	The username for your CloudBees account (e.g. your email address).
Password	The password for your CloudBees user account.

Cloud Foundry

Specify your Cloud Foundry user account details and related settings. For additional information, see the Cloud Foundry documentation .

Item	Description
Name	The name of the Cloud Foundry configuration.
Version	The Cloud Foundry version that you are using (a.k.a. the Service Broker API version): - V1 for version 1. - V2 for version 2.
Username (email)	Your username for your Cloud Foundry instance (e.g. your email address).
Password	Your password for your Cloud Foundry instance.
Provider (API URL)	The URL of the Cloud Controller in your Cloud Foundry instance (a.k.a. API endpoint or target URL).
Trust self-signed certificates	For version 2 (V2): Select this check box for self-signed cloud authentication certificates to be trusted. Otherwise, the self-signed certificates are rejected.
Organization	For version 2 (V2): The target organization (for deploying your applications).
Space	For version 2 (V2): The target space (within the organization).
Cloud instance	For version 1 (V1): One of the following: - Global for hosted Cloud Foundry. - Local (vcap.me) for a Cloud Foundry VCAP installation. - Micro - offline for Micro Cloud Foundry used in the offline mode.
Port	For a Cloud Foundry VCAP instance (Local (vcap.me)): The http port (`80` by default).
Domain	For a Micro Cloud Foundry instance (Micro - offline): The domain name associated with your Micro Cloud Foundry instance.

Google App Engine

Specify your Google App Engine Cloud user account details and related settings. For additional information, see the Google App Engine Cloud documentation .

Item	Description
Name	The name of the Google App Engine configuration.
Use passwordless login via OAuth2	Select this option to use the OAuth 2.0 authentication.
Login with email and password	Select this option to log in to the cloud using your email and password.
Email	Your Google email address.
Password	Your password.
Remember password	Select this check box if you want IntelliJ IDEA to remember your password (e.g. to save it in the password database). Note that IntelliJ IDEA won't remember the password if Do not remember passwords is currently selected on the Passwords page .

Heroku

Specify your Heroku user account details and related settings. For additional information, see the Heroku documentation .

Item	Description
Name	The name of the Heroku configuration.
Username (email)	The username for your Heroku account (e.g. your email address).
Password	The password for your Heroku user account.
Upload Public SSH Key	To be able to use the cloud, you should upload your public SSH key to it. (This isn't necessary if you did that earlier.) Click the link, click … in the Upload Public SSH Key dialog, and select your public SSH key file (`.pub`) in the dialog that opens.

OpenShift

Specify your OpenShift user account details and related settings. For additional information, see the OpenShift documentation .

Item	Description
Name	The name of the OpenShift configuration.
Server (API URL)	One of the following: – OpenShift (openshift.redhat.com) for OpenShift Online, the hosted PaaS service in the public cloud. – OpenShift Origin (broker.openshift.local) for OpenShift installed on your local computer or on your company network.
Domain	Your OpenShift domain.
Upload Public SSH Key	To be able to use the cloud, you should upload your public SSH key to it. (This isn't necessary if you did that earlier.) Click the link, click … in the Upload Public SSH Key dialog, and select your public SSH key file (`.pub`) in the dialog that opens.

Docker

For Docker configurations to be available, the Docker integration plugin must be installed.

Specify the settings for accessing the Docker Remote API and Docker Compose.

Item	Description
Name	The name of the configuration.
API URL	The Docker Remote API URL. The default is `https://192.168.99.100:2376` .
Certificates folder	The path to the certificates folder. (The certificates are generated automatically when you start the Docker daemon for the first time.) The default is `<your_home_directory>\.docker\machine\machines\default` .
Docker Compose executable	`docker-compose` or an actual path to `docker-compose.exe` (normally located in the Docker Toolbox installation folder). The default setting `docker-compose` for Docker Compose executable is fine if: – The actual name of the executable file is `docker-compose` . – The path to the directory where the file is located is included in the environment variable `Path` . See Docker Compose .
Import credentials from Docker Machine	If you want to import credentials for accessing the Docker Remote API from Docker Machine , select the check box and specify the associated settings: – Docker Machine executable. `docker-machine` or an actual path to `docker-machine.exe` (normally located in the Docker Toolbox installation folder). The default setting `docker-machine` is fine if: – The actual name of the executable file is `docker-machine` . – The path to the directory where the file is located is included in the environment variable `Path` . Detect. When you click Detect, IntelliJ IDEA sets Docker Machine executable to `docker-machine` , tries to detect the executable file and, if a success, shows its version. – Machine. After specifying the Docker Machine executable or clicking Detect, the list contains the Docker Machines available locally, and you can select the one to be used.

See also, Docker .

Toolbar

Use + to create configurations and – to delete them.

Coverage

File | Settings | Build, Execution, Deployment | Coverage for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Coverage for OS X

Ctrl+Alt+S

Use this page to define the way the coverage data will be processed.

Item	Description
When new coverage is gathered	
Show options before applying coverage to the editor	Choose this option to show the Code Coverage dialog every time you launch a new run configuration with code coverage. The coverage options dialog is displayed, when different coverage data have been produced.
Do not apply collected coverage	Choose this option to cancel applying the new code coverage results.
Replace active suites with the new one	Choose this option to have the active suites replaced with the new one every time you launch a new run configuration with code coverage.
Add to active suites	Choose this option to have the new code coverage suite added to the active suites every time you launch a new run configuration with code coverage.
Activate Coverage View	Select this check box to have the Coverage tool window opened automatically when an application or test is run with coverage.

Docker Registry

File | Settings | Build, Execution, Deployment | Docker Registry for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Docker Registry for OS X

For this page to be available, the Docker integration plugin must be installed.

This page lets you manage your Docker Registry configurations which represent your Docker image repository user accounts.

Toolbar

Icon	Description
+	Create a new configuration.
−	Delete the selected configurations.

Configuration settings

Item	Description
Name	The name of the configuration
Address	The image repository service URL, e.g. – registry.hub.docker.com for Docker Hub – quay.io for Quay
Username	The user name for your user account.
Password	Your password.
Email	The email address that you specified when creating your user account.
Server	The associated Docker Remote API configuration (used to connect to the service to check that your user account settings are correct).

Required Plugins

Use this page to specify plugins that are required for your project. In this case IntelliJ IDEA ensures that the plugins that are necessary for your project to work will be available.

Item	Description
(Alt+Insert)	Click this icon to add a required plugin for your project. In the dialog that opens, specify the following options: – Plugin - use this drop-down list to select a plugin that you want to add as required. – Minimum version - use this field to specify the minimum version of the selected plugin. – Maximum version - use this field to specify the maximum version of the selected plugin.
(Enter)	Click this icon to change the specified plugin or edit its version.
(Alt+Delete)	Click this icon to remove the specified plugin.

Web Services

File | Settings | Build, Execution, Deployment | Web Services for Windows and Linux

IntelliJ IDEA | Preferences | Build, Execution, Deployment | Web Services for OS X

`Ctrl+Alt+S`

Use this dialog box to specify the default server name and port for generating URL addresses of Web services, as well as the paths to installation directories of Web service engines that cannot be enabled directly via dedicated facets . These settings are used to enable support of these WS engines through the Enable Web Services Support dialog box.

Item	Description
External engines	In this section, specify the paths to the installation directories of external Web service engines. Enter the path manually, or click the Browse button ... and select the path in the dialog that opens. The following external engines are supported: – Glassfish / JAX-WS 2.2 RI / Metro 1.X / JWSDP 2.2 – Apache Axis 2 – CXF – JBoss – Xml Beans – WebSphere 6.x
Server name	In this text box, specify the default name of the server to be used in URL addresses. IntelliJ IDEA suggests `localhost` .
Server port	In this text box, specify the default port number to be used in URL addresses. IntelliJ IDEA suggests `8080` .
Prefix path for web services URL	In this text box, specify the default prefix to be used in URL addresses of Web services. IntelliJ IDEA suggests `/services` .
Maximum VM heap size when launching tools (Mb)	In this text box, specify the default maximum heap size in Mbs that will be used when a Web service is launched.

Languages and Frameworks

File | Settings | Languages and Frameworks for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks for OS X

When you select the Languages and Frameworks category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- JavaScript
- Play Configuration
- Schemas and DTDs
- ColdFusion
- Compass
- JSON Schema
- JavaFX
- Node.js and NPM
- OSGi
- OSGi Framework Instances
- Scala Compile Server
- SQL Dialects
- Play2
- Template Data Languages
- TypeScript
- Web Contexts
- XSLT
- XSLT File Associations
- PHP
- WordPress

JavaScript

File | Settings | Languages and Frameworks | JavaScript for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript for OS X

Ctrl+Alt+S

Use the pages in this section to specify the JavaScript language version for your project and configure JavaScript support in it.

In this part:

- JavaScript. Libraries
  - JavaScript. Usage Scope
- Code Quality Tools
  - JSLint
  - JSHint
  - Closure Linter
  - ESLint
  - JSCS
- Templates
- Bower
- Yeoman
- Meteor
- PhoneGap/Cordova

Item	Description
JavaScript Language Version	From this drop-down list, choose the JavaScript engine implementation to use in your project in accordance with the targeted environment. When you select an item in the list, the icons on the right of the field indicate the browsers that suit the selected JavaScript version, that is, support the features of the selected JavaScript engine implementation. The available options are: – JavaScript 1.8.5 – ECMAScript 3 – ECMAScript 5.1 – ECMAScript 6 – JSX Harmony – Flow
Prefer Strict mode	Select this check box to have the strict mode standard applied to JavaScript code. This helps improve your code by enforcing best practices and suppressing insecure ones. You can also configure the type of quotes to be used in `use strict` statements by choosing Double quotes or Single quotes from the String literal type drop-down list on the Code Style. JavaScript page of the Settings dialog box.
Weaker type guess for completion	– When this check box is cleared, the completion list contains only relevant variants. – When the check box is selected, the completion list contains additional variants in complex cases.

JavaScript. Libraries

File | Settings | Languages and Frameworks | JavaScript | Libraries for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Libraries for OS X

Ctrl+Alt+S

Use this page to set up additional JavaScript libraries to expand the basic assistance provided through the JavaScript plugin . Note that the JavaScript libraries are global.

In this section:

– Libraries
– Buttons
– New Library / Edit Library Dialog Box

Libraries

Item	Description
Enabled	Each check box in this column shows whether the corresponding library is avaiable or not: – A check box is selected when the corresponding library is defined and enabled in the current project. – A check box is marked with a Dash (`-`) if the corresponding library is defined and enabled on a more detailed level, for example, in a directory or in a file. – A check box is cleared when the corresponding library is disabled.
Name	This column shows the names of the directories defined for a project.
Type	This column shows library types.

Buttons

Item	Description
Add	Click this button to create a new JavaScript library in the New Library dialog box. Refer to the section Configuring JavaScript Libraries .
Edit	Click this button to change the name and contents of the selected library in the Edit Library dialog box.
Remove	Click this button to delete the selected library.
Download	Click this button to open the Download Library dialog box, where you can have IntelliJ IDEA download and install one of the popular official JavaScript-related libraries, such as: – Dojo – ExtJS – jQuery – jQuery UI – Prototype – etc. Besides the above listed official libraries, you can download stubs for TypeScript definition files . Choose the group of libraries in the drop-down list. The available options are Official libraries and TypeScript community stubs. Depending on your choice, IntelliJ IDEA displays a list of available libraries. Select the one to be downloaded and installed, and click Download and Install. You return to the JavaScript Libraries page where the new library is added to the list. CLick OK to save the settings.
Manage scopes	Click this button to configure libraries to be used for specific files and/or directories in the JavaScript. Usage Scope dialog box.

New Library / Edit Library Dialog Box

Item	Description
Name	Specify the library name.
Framework type	From this drop-down list, choose the framework to configure as a library.
Version	In this text box, specify the version of the selected framework to use.
Visibility	In this area, specify where you want the library to be available for associating with files and folders. The available options are: – Current project: when this option is chosen, the library can be associated with files and folders within the current project only. If you later try to use the framework with another project, you will have to configure the library anew. – Global: choose this option to enable associating the library with any of your IntelliJ IDEA projects.
Files	In this section, set up the library contents.
+ (Add)	Click this button to attach a JavaScript file or directory from the file system.
– (Remove)	Click this button to detach the selected file or directory from a library.
Name	This read-only column shows the name of the selected library file or the names of relevant library files from the selected directory.
Type	Click the column to show the drop-down list of the available versions of library files or directories: debug or release.

IntelliJ IDEA enables you to create a library containing just one `.js` file, if this file is located on the Internet and can be accessed over HTTP. If you refer to a JavaScript library that is not yet available locally, but is available online, use the Download library intention action :

The library will be placed to the user home directory, and will appear in the list of configured libraries in the JavaScript - Libraries page of the Settings dialog.

JavaScript. Usage Scope

File | Settings | JavaScript | Libraries - Manage Scopes for Windows and Linux

IntelliJ IDEA | Preferences | JavaScript | Libraries - Manage Scopes for OS X

Ctrl+Alt+S

The dialog box opens when you click the Manage Scopes button in the JavaScript. Libraries page.

Use this dialog box to define the scopes where the JavaScript libraries should apply for code completion, highlighting and navigation. The scopes may cover the whole project, directories and even individual files. This helps make JavaScript code completion more precise, and avoid too long suggestion lists.

Item	Description
File/Directory	This column displays the project tree view.
Library	This column displays libraries for a file or directory, if applicable. If a library can be specified for a certain node of the project tree view, click the Library column for a selected file or directory, and choose the desired library from the list of available libraries. If JavaScript libraries are not applicable to a particular node, 'N/A' is shown in grey font.

Code Quality Tools

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools for OS X

Ctrl+Alt+S

Use the pages under this node to enable the built-in JavaScript code verifiers and configure their behaviour.

In this part:

- JSLint
- JSHint
- Closure Linter
- ESLint
- JSCS

JSLint

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools | JSLint for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools | JSLint for OS X

`Ctrl+Alt+S`

Use this page to enable the built-in JavaScript JSLint code verifier and configure its behaviour and sensitivity.

Item	Description
Enable	Select this check box to have JSLint applied to verify the code in the current project. After that the other controls in the page are enabled.
Tolerate	In this area, specify the discrepancies you want JSLint to consider worth reporting by enabling or disabling JSLint options. By default, discrepancies of all types are considered worth reporting. To have the tool skip discrepancies of a certain type, select the check box next to this type.
Assume	In this area, specify for which environments you want global properties predefined.
Stop on first error	Select this check box to have the verification process suspend as soon as the first valuable discrepancy is detected.
Safe Subset	Select this check box to have ADSafe safe subset rules enforced.
Verify ADsafe	Select this check box to have the ADSafe rules enforced.
Indentation	In this tet box, type the number of spaces to be used for indentation. This setting corresponds to the `indent` option.
Maximum line length	In this text box, limit the number of characters acceptable in one line. This setting corresponds to the `maxlen` option.
Maximum number of errors	In this text box, specify the maximum number of warnings that can be reported. The default setting is 50. This setting corresponds to the `maxerr` option.
Predefined	In this text box, specify the predefined global variables by names or through object keys. Use commas (`,` > as separators. Note This setting corresponds to the `predef` option.
Validate also	In this area, enable or disable additional verification of HTML, CSS, or JSON context and configure the checking mode, when enabled. – HTML: select this check box to have HTML context verified. Specify how you want HTML event handlers and HTML fragments treated: by default, they are reported as errors. To have the tool skip them, select the Tolerate HTML event handlers and Tolerate HTML fragments check boxes respectively. – CSS: select this check box to have CSS context verified. Specify how you want CSS workarounds treated: by default, they are reported as errors. To have the tool skip them, select the Tolerate CSS workarounds check box. – JSON: select this check box to have HTML context verified.

JSHint

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools | JSHint

Use this page to enable the built-in or downloaded JavaScript JSHint code verifier and configure its behaviour and sensitivity.

Item	Description
Enable	Select this check box to have JSHint applied to verify the code in the current project. After that the other controls in the page are enabled.
Use config files	Select this check box to have the code verified according to the settings from a configuration file. A configuration file is a JSON file with the extension `.jshintrc` that specifies which JSHint options should be enabled or disabled. IntelliJ IDEA will look for a `.jshintrc` file in the working directory. If the search fails, IntelliJ IDEA will search in the parent folder, then again in the parent folder. The process is repeated until IntelliJ IDEA finds a `.jshintrc` or reaches the project root. To have IntelliJ IDEA still run verification in this case, specify the default configuration file to use. Starting with IntelliJ IDEA version 6.0.1, the globals setting in `.jshintrc` is supported. For earlier versions, a Predefined (, separated)" tree node is displayed in the bottom of the tree in the UI. When this check box is selected, the Options area is hidden and the default verification settings are unavailable.
Version	Use this drop-down list to choose the version of the tool to apply. IntelliJ IDEA comes bundled with version 1.0.0, which is used by default. IntelliJ IDEA provides the ability to download another version, which is not bundled. Actually, the alternative version is downloaded only once, whereupon it is available without download.
Options	In this area, configure JSHint behaviour by enabling or disabling JSHint options. To enable or disable an option, select or clear the corresponding check box respectively. The controls in the area fall into two groups: – Enforcing options: select the check boxes in this group to enable very strict behaviour of the verification tool and thus allow only safe JavaScript. – Relaxing options: select/clear the check boxes in this area to suppress warnings when certain types of discrepancies are detected. – Environments: select/clear these check boxes to specify for which environments you want global properties predefined. The area is available only when the Use config files

Closure Linter

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools | Closure Linter for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools | Closure Linter for OS X

`Ctrl+Alt+S`

Use this page to enable the JavaScript Closure Linter code verifier and to configure its behaviour and sensitivity.

Item	Description
Enable	Select this check box to have Closure Linter applied to verify the code in the current project. After that the other controls in the page are enabled.
Closure Linter executable file	In this text box specify the path to the Closure Linter executable file: – `<Python_home>\Scripts\jslint.exe` for Windows – `/usr/local/bin/gjslint` for Linux and OS X
Configuration file	In this text box, specify the location of the previously created configuration text file to apply.

ESLint

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools | ESLint for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools | ESLint for OS X

Ctrl+Alt+S

Use this page to enable the JavaScript ESLint code verifier and to configure its behaviour and sensitivity.

Item	Description
Enable	Select this check box to have ESLint applied to verify the code in the current project. After that the other controls in the page are enabled.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
ESLint Package	In this field, specify the location of the `eslint` package installed in the current project, see Installing ESLint .
Configuration file	In this area, appoint the configuration to use. By default, IntelliJ IDEA first looks for a `.eslintrc` configuration file. IntelliJ IDEA starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. If no `.eslintrc` file is found, ESLint uses its default embedded configuration file. Accordingly, you have to define the configuration to apply either in a `.eslintrc` configuration file, or in a custom JSON configuration file, or rely on the default embedded configuration. – To have IntelliJ IDEA look for a `.eslintrc` file, choose the Search for .eslintrc option. If no `.eslintrc` file is found, the default embedded configuration file will be used. – To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the ... button and select the relevant file from the dialog box that opens.
Additional Rules Directory	In this field, specify the location of the files with additional code verification rules. These rules will be applied after the rules from `.eslintrc` or the above specified custom configuration file and accordingly will override them.
Extra ESLint Options	In this field, specify additional command line options to run ESLint with using spaces as separators. See ESLint Command Line Interface Options for details.

JSCS

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools | JSCS for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools | JSCS for OS X

`Ctrl+Alt+S`

Use this page to enable the JavaScript JSCS code verifier and to configure its behaviour and sensitivity.

Item	Description
Enable	Select this check box to have JSCS applied to verify the code in the current project. After that the other controls in the page are enabled.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the … button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
JSCS Package	In this field, specify the location of the `jscs` package installed in the current project, see Installing JSCS .
Configuration File	In this area, appoint the configuration to use. By default, IntelliJ IDEA first looks for a `jscsConfig` property in the `package.json` file of the current project. If no such property is found, IntelliJ IDEA looks for a `.jscsrc` or a `.jscs.json` configuration file. IntelliJ IDEA starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. Accordingly, you have to define the configuration to apply either as a `jscsConfig` property in the `package.json` file or in a `.jscsrc` or a `.jscs.json` configuration file, or in a custom JSON configuration file. You can also apply a predefined set of rules, either independently or in combination with a configuration file. In the latter case, the rules from the configuration file override the predefined rules. – To have IntelliJ IDEA look for a `jscsConfig` property in the `package.json` file or for a `.jscsrc` or a `.jscs.json` file, choose the Search for config(s) option. – To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the … button and select the relevant file from the dialog box that opens. – To have a predefined set or rules applied, choose the desired set from the Code Style Preset drop-down list.
Code Style Preset	From this drop-down list, choose the set of predefined rules associated with the code style you use.

Templates

File | Settings | JavaScript - Templates for Windows and Linux

for OS X

Ctrl+Alt+S

On this page, configure processing of EJS inclusions, Handlebars expressions, and Mustache templates in IntelliJ IDEA.

This page only appears in the Settings/Preferences dialog, when the corresponding plugin is downloaded and enabled.

Item	Description
Handlebars/Mustache	In this area, configure processing of Handlebars expressions and Mustache templates in IntelliJ IDEA. The settings specified on this page apply to dedicated Handlebars and Mustache files that have the extension `.hbs` or `.mustache` respectively. – Automatically insert closing tag: – When this check box is selected, IntelliJ IDEA automatically inserts the second closing curly brace (`}`) of a Handlebars expression as soon as you type the first closing one. IntelliJ IDEA also recognizes triple stashes (`{{{`) that prevent escaping values inside expressions. In this case, IntelliJ IDEA automatically inserts two closing curly braces as soon as you type the first closing one. – When this check box is cleared, you have to type the closing curly braces and triple stashes manually. – Enable formatting: – Select this check box to have Handlebars expressions and Mustache templates automatically reformatted during code generation, refactoring, or reformatting (Ctrl+Alt+L). – Clear the check box to have the original formatting of Handlebars expressions and Mustache templates preserved. – Open HTML files as Handlebars/Mustache: – When this check box is selected, files with the `.html` extension are treated as Handlebars/Mustache files so IntelliJ IDEA recognizes and processes Handlebars expressions and Mustache templates. The extensions of files remain unchanged but file names are supplied with the icons. – When this check box is cleared only files with the `.hbs` extension are treated as Handlebars/Mustache files and Handlebars expressions and Mustache templates within them are recognized and processed. – Language for comments: From this drop-down list, select the language to inherit the style for comments from. When you enter a line or block comment by pressing Ctrl+Slash or Ctrl+Shift+Slash, IntelliJ IDEA inserts the comment delimiters that are used in the chosen language, for example, `{{!----}}` for Handlebars, `/**/` for JavaScript, `<!---->` for HTML, etc. For more details about Handlebars expressions and Mustache templates see http://handlebarsjs.com/ . The area is only available when the Handlebars/Mustache plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
EJS	In this area, specify delimiters for inclusions of EJS in your code. Based on these delimiters, IntelliJ IDEA will recognize and process such inclusions correctly. A pair of delimiters is also referred to as execute tag. – EJS open delimiter: In this text box, type the character string that will indicate the beginning of an EJS inclusion. The default delimiter is `<%` . – EJS close delimiter: In this text box, type the character string that will indicate the end of an EJS inclusion. The default delimiter is `%>` .

Bower

File | Settings | Languages and Frameworks | JavaScript | Bower

On this page, specify the location of the Bower package manager executable and configuration files and install, uninstall, or upgrade external tools through it.

Item	Description
Node interpreter	Specify here the location of the Node.js executable. Type the path manually, or choose it from the drop-down list if it was previously used, or click the Browse button and choose the location in the dialog box that opens. See Node.js for details.
Bower executable	In this field, specify the location of the Bower executable file (`bower.cmd` or other depending on the operating system used). Type the path manually, or choose it from the drop-down list if it was previously used, or click the Browse button and choose the location in the dialog box that opens.
bower.json	In this text bos, specify the location of the bower.json file. Type the path manually, or click the Browse button and choose the location in the dialog box that opens.
Dependencies	The area shows a list of all the -dependent packages that are currently installed on your computer. – Package: this read-only field shows the name of a package, exactly as it should be referenced if you were installing it in the command line mode. – Version: this read-only field shows the version of the package installed on your computer. – Latest: this read-only field shows the latest released version of the package. If a package is not up-to-date, it is marked with a blue arrow. – Click + to have a new package installed. In the Available Packages dialog box that opens, select the relevant package. Click Install Package when ready. – Click – to have the selected package removed. – Click ↑ to have the current version of the selected package replaced with the latest released version. The button is enabled only when the selected project is not up-to-date.

Yeoman

File | Settings | Languages and Frameworks | JavaScript | Yeoman for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Yeoman for OS X

`Ctrl+Alt+S`

The page is available only when the Yeoman and the NodeJS plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

On this page, specify the location of the Node.js executable file and the Yeoman project generator package installed through Node Package Manager.

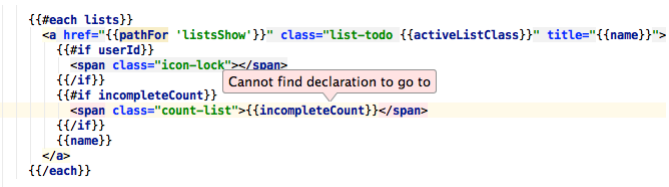
Item	Description
Node interpreter	Specify here the location of the Node.js executable. Type the path manually, or choose it from the drop-down list if it was previously used, or click the Browse button ... and choose the location in the dialog box that opens. See Node.js for details.
Yeoman yo package	In this field, specify the location of the global `yo` package, see Installing and Removing External Software Using Node Package Manager , section Installing an External Tool Globally.

Meteor

File | Settings | Languages and Frameworks | JavaScript | Meteor for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Meteor for OS X

The page is available when the Meteor plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Item	Description
Meteor executable	In this field, specify the location of the Meteor executable file (see Installing Meteor).
Automatically exclude ".meteor/local" directory on open project	– By default, the check box is selected and the `.meteor/local` folder, which is intended for storing the built application, is automatically marked as excluded and is not involved in indexing. – Clear the check box to have IntelliJ IDEA show the `.meteor/local` folder and its contents in the Project tool window. By default, excluded files are shown in the project tree. To have the `.meteor/local` folder hidden, click the ⚙ button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.
Enable Meteor 'Hot code push'	– Select this check box to activate the native Meteor hot code pushes functionality and thus have the changes made to have the client-side code during a debugging session applied. By default, the check box is selected. Changes made to the server-side code are uploaded through the IntelliJ IDEA Live Edit functionality, see Configuring the Update Policy for the Server Side Code . – When the check box is cleared, IntelliJ IDEA does not provide any way to apply the changes made to the client-side code during a debugging session.
Automatically import Meteor packages as external library	– When the check box is selected, IntelliJ IDEA automatically imports the external packages from the `meteor/packages` file. As a result, IntelliJ IDEA provides full range coding assistance: resolves references to Meteor built-in functions, for example, `check(true)` , and to functions from third-party packages, provides proper syntax and error highlighting, supports debugging with source maps, etc. – When this check box is cleared, IntelliJ IDEA does not automatically import the external packages from the `meteor/packages` file. As a result no coding assistance is provided. To improve the situation, open the `meteor/packages` file in the editor and click the Import packages as library link or run the `meteor --update` command. By default, the check box is selected.
Weak search for Spacebars templates	– Select this check box to enable IntelliJ IDEA to search inside Spacebars templates. When this check box is selected and you invoke the Go to Declaration action on a helper, IntelliJ IDEA displays a list of all occurrences of this helper in templates. Choose the relevant one from the list. – When you invoke Got to Declaration with this check box cleared, IntelliJ IDEA does not perform any search and displays a tooltip which the following message: Cannot find declaration to go to. 

PhoneGap/Cordova

File | Settings | Languages and Frameworks | JavaScript | PhoneGap/Cordova for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | PhoneGap/Cordova for OS X

Ctrl+Alt+S

The page is available when the PhoneGap/Cordova plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Item	Description
PhoneGap/Cordova Executable Path	In this field, specify the location of the executable file `phonegap.cmd`, `cordova.cmd`, or `ionic.cml` (see Installing PhoneGap/Cordova/Ionic).
PhoneGap/Cordova Version	This read-only field shows the installed PhoneGap/Cordova/Ionic version that IntelliJ IDEA detects automatically.
PhoneGap/Cordova Working Directory	In this field, specify the folder under which the PhoneGap/Cordova/Ionic application files to run are stored.
Automatically exclude 'platforms' directory on open project	Select this check box to have the platforms directory marked as excluded automatically when you open the project. As a result, IntelliJ IDEA ignores it during indexing, parsing, and code completion.
Plugins	In this area, configure a list of plugins to use in your development by installing required packages. The list shows all the PhoneGap/Cordova/Ionic plugins that are currently installed on your computer, both at the global and at the project level. – To install a plugin, click the Install button +. In the Available Packages dialog box that opens, select the required package. To have the plugin installed globally so it is accessible from all your IntelliJ IDEA projects, select the Options check box and type `--global` in the text box. Click Install Package. – To remove a plugin, select it in the list and click the Uninstall button —. – To upgrade a plugin to the latest available version, select the plugin in the list and click the Upgrade button ↑. See Apache Cordova Plugins and PhoneGap Plugins for information about plugins and their use.

Play Configuration

File | Settings | Languages and Frameworks | Play Configuration

Note For this page to be available, the Playframework Support plugin must be enabled. See Enabling and Disabling Plugins .

Use this page for specifying the Play framework settings such as the Play framework installation directory and the working directory for the `play` command-line utility (the Play console).

Item	Description
Download	Click this link to open the Play framework downloads page which lets you select and download the necessary version of the Play framework. (See which versions are supported by IntelliJ IDEA.)
Home	Specify the Play framework installation directory. Type the path in the field, or click … (`Shift+Enter`) and select the directory in the dialog that opens .
Working directory	Specify the directory from which commands of the `play` command-line utility are to be run. Usually, this is a root directory of your Play application. Type the path in the field, or click … (`Shift+Enter`) and select the directory in the dialog that opens .
Show on console run	Select this check box to be able to see and modify the Play framework settings discussed above each time you access the `play` command-line utility in IntelliJ IDEA.

Schemas and DTDs

File | Settings | Languages and Frameworks | Schemas and DTDs for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | Schemas and DTDs for OS X

`Ctrl+Alt+S`

The settings on this page define how your XML, HTML and XHTML files are validated.

– External Schemas and DTDs

– Ignored Schemas and DTDs

External Schemas and DTDs

Local XML schema (XSD) and DTD files that are used to validate your XML files are listed in this section.

Each entry is a mapping of a URI that may be referenced in your XML file onto an appropriate local schema or DTD file. Use +, – and to create, remove and edit the mappings. See also, Referencing XML Schemas and DTDs .

Item	Keyboard Shortcut	Description
URI		A URL of an XML schema (XSD) or DTD file (e.g. `http://www.example.org/xsds/example.xsd`) or a namespace URI (e.g. `http://www.example.org`, `urn:jboss:domain:1.0`). Readonly. If an XML file references the specified URI, it's validated according to a local file whose path is shown in the Location columns.
Location		Path to corresponding local XSD or DTD file (readonly).
Project		If a check box is not selected, a mapping is available in all of your projects. Select the mappings that you want to be available only in the current project. The selected mappings are stored in `.idea/misc.xml` or the project `.ipr` file. These files are normally shared between development team members through version control.
+	`Alt+Insert`	Use this icon or shortcut to open the Map External Resource Dialog to define a new mapping between a URI and a local file.
–	`Alt+Delete`	Use this icon or shortcut to remove the selected mappings from the list.
	`Enter`	Use this icon or shortcut to edit the selected mapping. (The Map External Resource Dialog dialog will open.)

Ignored Schemas and DTDs

URIs for ignored schemas and DTDs are listed in this section. (If an XML file references a URI listed in this section, IntelliJ IDEA ignores that URI and doesn't mark it as an error in the editor.) Use +, – and to add, remove and edit the URIs. See also, Referencing XML Schemas and DTDs .

Item	Keyboard Shortcut	Description
+	`Alt+Insert`	Use this icon or shortcut to add a new URI to the list.
	`Enter`	Use this icon or shortcut to edit the selected URI.
–	`Alt+Delete`	Use this icon or shortcut to remove the selected URIs from the list. The corresponding URIs become available for mapping to local XSD or DTD files.

Default XML Schemas

File | Settings | Languages and Frameworks | Schemas and DTDs | Default XML Schema for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | Schemas and DTDs | Default XML Schema for OS X

`Ctrl+Alt+S`

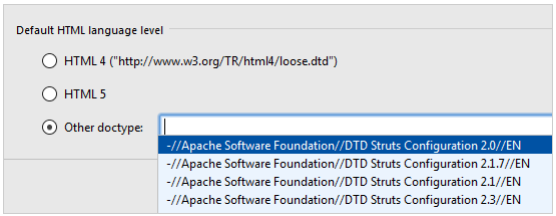
On this page:

– Default HTML Language Level
– Default XML Schema Version

Default HTML Language Level

Normally, an HTML or an XHTML file has the `<!DOCTYPE>` declaration which states the language level of the used in the source code from the file. This language level is used as a standard against which the contents of the file are validated. If an HTML or an XHTML file does not have a `<!DOCTYPE>` declaration, the contents of the file will be validated against the default standard (schema). In the Default HTML Language Level area, choose the default schema to validate HTML and XHTML files without a `<!DOCTYPE>` declaration. The available options are:

– HTML 4 or HTML 5: Choose one of these options to have files treated as HTML 4 or HTML 5 and validated against one of these standards.
– Other doctype: Choose this option to have HTML files by default validated agains a custom DTD or schema and specify the URL of the DTD or schema to be used. Note that code completion is available in this field: press `Ctrl+Space` to see the list of suggested URLs.



Default XML Schema Version

In this area, choose the XSD (XML Schema Definition) Schema to validate XML files. The available options are:

– XML Schema 1.1 See W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures and W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes for details.
– XML Schema 1.0 See XML Schema Part 1: Structures Second Edition and XML Schema Part 2: Datatypes Second Edition for details.

XML Catalog

File | Settings | Languages and Frameworks | Schemas and DTDs | XML Catalog for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | Schemas and DTDs | XML Catalog for OS X

Ctrl+Alt+S

Use this dialog to configure your .properties file. This file contains lists of catalog.xml files. The catalog.xml files contain the information on how to resolve various PUBLIC and SYSTEM identifiers during XML processing.

Item	Description
Catalog property file	Use this field to specify the location of the .propeties file. Type the path in the field, or click ... and select the required file in the dialog that opens . For more detailed description of the properties in the catalog property file, see the example of Annotated CatalogManager properties file .

ColdFusion

File | Settings | Languages and Frameworks | ColdFusion for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | ColdFusion for OS X

Ctrl+Alt+S

Use this page to select the ColdFusion language version and to specify mappings for deploying your application to the ColdFusion server.

Item	Description
Language Version	Select the ColdFusion language version.
Info	Use this area to specify a ColdFusion debugger server information. Before you specify ColdFusion debugger server port in IntelliJ IDEA, you need to manually set a debugger server port and manually start a debugger server in ColdFusion Administrator . – User Name - specify a user name that you use to access ColdFusion Administrator . – Debugger Server Port - specify a debugger server port that you have set with JVM argument `-DDEBUGGER_SERVER_PORT=port` on the Java and JVM page of the ColdFusion Administrator .
Server Mappings	In this area, set the correspondence between local folders and logical paths on the server. – Directory Path - specify the folder on your local computer, where the source code and resources to be deployed are stored. Type the path manually or click … and choose the folder in the dialog box, that opens. – Logical Path - specify the target folder to deploy the application to. The path should be specified relative to the server root URL defined in the corresponding run configuration . – + (Alt+Insert) - use this icon or shortcut to create a new mapping. – – (Alt+Delete) - use this icon or shortcut to remove the selected item from the list.

Compass

File | Settings | Languages and Frameworks | Compass for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | Compass for OS X

`Ctrl+Alt+S`

Use this page to integrate the Compass framework with IntelliJ IDEA and thus enable transpilation of Sass and SCSS files from Compass-specific projects into CSS. For details, see Using File Watchers .

Prerequisites

Before you start working with Compass, make sure that the Sass support plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Make sure that Ruby is installed on your computer.

For more details, see Working with Sass and SCSS in Compass Projects .

Item	Description
Enable Compass support	Select this check box to activate Compass support including the possibility to transpile Sass and SCSS files from your Compass project into CSS.
Compass executable file	In this field, specify the location of the `compass` executable file under the Ruby installation. Type the path manually, for example, `C:\Ruby200-x64\bin\compass`, or choose it from the drop-down list, or click the Browse button … and choose the location of the `compass` file in the dialog box that opens.
Config path	In this field, specify the location of the project Compass configuration file `config.rb`. Type the path manually, for example, `C:\my_projects\\compass_project\config.rb`, or choose it from the drop-down list, or click the Browse button … and choose the location of the `compass` file int he dialog box that opens. The Compass configuration file `config.rb` is generated during project set-up through `compass create` or `compass init` commands.

JSON Schema

File | Settings | Languages and Frameworks | JSON Schema for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JSON Schema for OS X

Ctrl+Alt+S

On this page, create a list of your custom JSON schemas in addition to the system schemas provided by IntelliJ IDEA and configure the scope for each schema, that is, specify the `.json` files to be validated against it. The page consists of two panes: the List of Schemas pane and the Schema Details pane.

On this page:

– List of Schemas Pane
– Schema Details Pane
– Handling Conflicts Among Scopes of Schemas

List of Schemas Pane

The central pane of the page shows a list of custom JSON schemas against which `.json` files in the current project will be validated. Based on a schema, IntelliJ IDEA checks the structure of `.json` configuration files, reports errors (for example, informs you about missing mandatory properties), provides code completion and documentation look-up.

The pane shows only custom schemas that you have previously downloaded or created yourself, in either case, a custom schema must meet the JSON schema standards and must be located under the project root.

System schemas that IntelliJ IDEA provides for all supported frameworks and technologies are not shown in the list.

– To add a schema to the list, click + on the toolbar of the pane and select the relevant schema file under the project root in the dialog box that opens.
– To remove a schema from the list, select it and click – on the toolbar of the pane.
– To configure the scope of a schema, select the schema and specify the files to be validated against it in the right-hand pane of the page.

Schema Details Pane

The pane shows the details of the schema selected in the List of Schemas pane: the name of the selected schema, the `.json` file that implements it, and a list of files and folders that are validated against the schema.

The list contains the names of specific files, the names of entire directories, and filename patterns. You do not need to specify full paths to files and folders, IntelliJ IDEA searches for files and folders with the specified names and the search is restricted to the the current project.

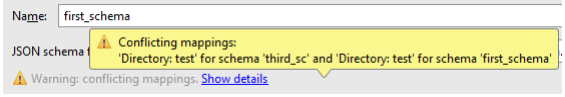
Based on the list, IntelliJ IDEA internally creates a list of files to be validated. Each file is included in the list only once: if a file with the specified name is stored in a directory from the list or its name matches a pattern, the file is still validated only once.

– To add new files or folders to the list, click + on the toolbar of the pane and specify the file, folder, or a file pattern in the Add JSON Schema Mapping Dialog that opens.
– To remove an item from the list, select it and click – on the toolbar of the pane.

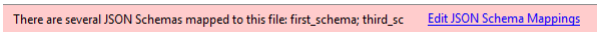
Handling Conflicts Among Scopes of Schemas

A conflict arises when a file, or a folder, or a pattern belongs to the scopes of two or more schemas. IntelliJ IDEA analyzes scopes in two modes:

– Static Analysis detects conflicts in scopes of custom schemas. If a conflict is detected, IntelliJ IDEA displays a warning in the Schema Details pane. To view the overlapping scopes, click the Show details link. IntelliJ IDEA shows a pop-up message where the conflicting scopes and schemas are listed:



– Dynamic Analysis detects conflicts in scopes of both system and custom schemas. This type of analysis is started when you open a file that belongs to a certain scope. If a conflict is detected, IntelliJ IDEA displays a warning at the top of the editor tab:



Click the link to open the JSON Schema page and edit the scope of the conflicting custom schema. Note that you cannot edit the scope of system schemas.

Add JSON Schema Mapping Dialog

File | Settings | Languages and Frameworks | JSON Schema for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JSON Schema for OS X

The dialog box opens when you select a `.json` schema in the list on the JSON Schema page and click + in the right-hand pane. In this dialog, specify the `.json` files to be validated against the selected schema.

Item	Description
Files Under	Choose this option to add files stored in a specific folder name to validation. Type the name of the folder manually or click ... and choose the folder in the dialog box that opens.
Filename pattern	Choose this option to have all files with the names that match a pattern validated.
File	Choose this option to add files with a specific name to validation. Type the filename manually or click ... and choose the file in the dialog box that opens.

JavaFX

File | Settings | Languages and Frameworks | JavaFX

Specify where the JavaFX Scene Builder executable file is located. If you do so, you'll be able to open your FXML files in the Scene Builder right from within IntelliJ IDEA.

Item	Description
Path to SceneBuilder	Specify the path to the JavaFX Scene Builder executable file. Type the path in the field, or click … (Shift+Enter) and select the Scene Builder executable file in the dialog that opens .

Node.js and NPM

File | Settings | Languages&Frameworks | Node.js and NPM for Windows and Linux

IntelliJ IDEA | Preferences | Languages&Frameworks | Node.js and NPM for OS X

Ctrl+Alt+S

This page appears in the Settings dialog box, when the Node.js plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Item	Description
Node interpreter	In this field, specify the local Node.js interpreter to use. Choose the interpreter from the drop-down list or click ... and choose the interpreter in the dialog box that opens. The term local Node.js interpreter denotes a Node.js installation on your computer. The term remote Node.js interpreter denotes a Node.js installation on a remote host or in a virtual environment set up in a Vagrant instance. On the Node.js and NPM page, you can specify only local interpreters. Remote interpreters are configured in the Configure Node.js Remote Interpreter Dialog dialog which can be accessed only from the Run/Debug Configuration: Node JS dialog. See Configuring Node.js Interpreters for details.
Version	This read-only field shows the current version of the runtime environment.
Code Assistance	When developing a Node.js application it can be convenient to have code completion, reference resolution, validation, and debugging capabilities for Node core modules (`fs` , `path` , `http` , etc.). However, these modules are compiled into the Node.js binary. IntelliJ IDEA provides the ability to configure these sources as a JavaScript library and associate it with your project. – If the Node.js core module sources are not set up, IntelliJ IDEA displays a notification Node.js Core Library is not enabled with an Enable button. Click this button to have IntelliJ IDEA configure Node.js Core sources automatically. When the configuration is completed, IntelliJ IDEA displays information about the currently configured version, the notification Node.js Core Library is enabled, and adds two buttons: the Disable button and the Usage scope button. – If the library is set up, IntelliJ IDEA displays information about the currently configured version, the notification Node.js Core Library is enabled, and adds two buttons: the Disable button and the Usage scope button. – Click the Disable button to discard the configuration of the Node.js Core libraries in the current project. – Click the Usage scope button to associate the desired directories with libraries. If necessary, configure the scope in which the Node.js Core sources are treated as libraries. Click the Usage scope button, and in the Usage Scope dialog box that opens, click the desired directories, and from the drop-down list select the newly configured Node.js core module sources library.
Packages	A number of tools are started through Node.js, for example, the CoffeeScript, TypeScript, and Less transpilers, YUI, UglifyJS, and Closure compressors, Karma test runner, Grunt task runner, etc. The Node Package Manager (npm) is the easiest way to install these tools, the more so that you have to install Node.js anyway. The Packages area shows a list of all the NPM-dependent packages that are currently installed on your computer. – Package: this read-only field shows the name of a package, exactly as it should be referenced if you were installing it in the command line mode. – Version: this read-only field shows the version of the package installed on your computer. – Latest: this read-only field shows the latest released version of the package. If a package is not up-to-date, it is marked with a blue arrow. – Click + to have a new package installed. In the Available Packages dialog box that opens, select the relevant package. To have the package installed globally, select the Options check box and type `-g` in the Options text box. Global installation makes the package available at the IntelliJ IDEA level so it can be used in any IntelliJ IDEA project. Click Install Package when ready. – Click – to have the selected package removed. – Click ↑ to have the current version of the selected package replaced with the latest released version. The button is enabled only when the selected project is not up-to-date.

OSGi

File | Settings | Languages and Frameworks | OSGi

Use this page to manage project specific and application wide settings of OSGi -based applications.

Prerequisites

This page appears in the Settings/Preferences dialog, when Osmorc plugin is enabled.

The plugin is bundled with IntelliJ IDEA and activated by default. If it is disabled, you can manually enable the plugin .

Item	Description
OSGi framework	From this drop-down list, select the desired framework. The list contains all the framework instances defined for the currently running IntelliJ IDEA at the IDE level. If the framework instance you need is missing, switch to the Framework Instances page of the IDE-level OSGi settings, and define the required instance there.
Default manifest	Select the path to the default `MANIFEST.MF` file. This option makes sense for the OSGi facets, where the bundle creation is performed using the existing manifest and facet configuration (the option Use existing manifest and bundle using facet configuration is turned on). If any other type of creating bundles is selected, this field is ignored.
Output path	Type here the path to the directory, where all compiled bundles will be stored, or click the browse button and locate the desired directory in the file system. This path is only used, if Jar output path of the OSGi facet is set to Place in project-wide OSGi output path. Otherwise, this option is ignored, and the bundles are generated to the module output, or facet-specific path.
Apply to all facets	Click this button to apply the specified output path to all OSGi facets in the current project. This button changes Jar output path for all facets; even though the option Place in module's output path has been selected in a facet, it will be changed to Place in project-wide OSGi output path.
Import Bnd/Bndtools projects automatically	Select this check box to import Bnd/Bndtools projects automatically if IntelliJ IDEA detects any changes in bnd files. Note that you can manually reimport Bnd/Bndtools project or module by selecting Reimport Projects or Reimport Workspace from the context menu in the Project tool window.

OSGi Framework Instances

File | Settings | Languages and Frameworks | OSGi Framework Instances

Prerequisites

This page appears in the Settings/Preferences dialog, when Osmorc plugin is enabled.

The plugin is bundled with IntelliJ IDEA and activated by default. If it is disabled, you can manually enable the plugin .

Use this page to add and remove IDE-level framework instance definitions for the currently running IntelliJ IDEA. The set of available framework definitions determine the contents of the OSGi framework drop-down list in the Project Settings tab.

Item	Description
Framework Instances	The field shows a list of all the framework instances that are defined for the currently running IntelliJ IDEA.
Type	This read-only field shows the type of the currently selected definition.
Home directory	This read-only field shows the location of the currently selected framework instance.
Name	This read-only field shows the name of the currently selected definition.
Version	This read-only field shows the version of framework the selected definition is based on.
(Alt+Insert)	Click this button to open the Add OSGI Framework drop-down list and select the type of a new framework instance that you want to configure. The specified OSGI Framework Instance dialog opens.
(Enter)	Click this button to open the FrameWork Instance dialog box and edit the definition of the selected framework there. The icon is active if there is at least one framework defined in the Framework Instances field.
(Alt+Delete)	Click this button to remove the selected definition from the list.
	Click these buttons to move up or to move down the selected framework instance.

OSGi Framework Instance Dialog

IntelliJ IDEA lets you use this dialog to configure OSGi framework instance. The OSGI Framework Instance dialog appears when you try to add a new OSGI framework instance from the Add OSGi Framework drop-down list, on the OSGi Framework Instances page.

Item	Description
Home Directory	Use this field to specify the location of your new OSGi framework.
Name	Use this field to specify the name of the framework. By default, this field contains a type of your OSGi framework that you are trying to add.
Version	Use this field to specify the version of the framework. As long as the home directory is specified this field is filled in automatically.

Library Bundling

File | Settings | Languages and Frameworks | OSGi Framework Instances | Library Bundling

Use this page to define rules that determine the libraries to be included in bundles and have manifest file entries generated based on these rules.

ItemDescription

Rules

In this area, configure a list of regular expressions that define the libraries to be included in a bundle.

Tip The rules are processed from top to bottom. If a library matches several regular expressions, the one that comes last is applied. Therefore it is recommended that you have the list sorted as follows: the more strict a rule the closer it is to the bottom. For example, it is recommended to put the pattern `*.*` at the top of the list and an exact match in the bottom.

ItemCommandDescription

+	Add	Click this button to have a new entry added to the list.
−	Remove	Click this button to remove the selected entry from the list.
	Copy	Click this button to create a copy of the selected rule.
	Move Up and Move Down	Use these buttons to change the order of rules in the list.

Library name pattern

In this text box, specify the regular expression that constitutes the entry selected in the Rules list box.

Note When a new rule is added to the list, the text box by default shows `.*`.

Manifest entries

This read-only field displays the manifest file entries generated on the basis of specified rules. The entries are sorted in accordance with the order of rules in the Rules list box.

Tip The Add New line quick fix is available.

Never bundle

If this check box is selected, the libraries that match the currently selected rule are never bundled.

Process no further rules

Select this check box to suppress processing rules that come in the Rules list box after the selected rule.

Scala

Use this page to configure Scala settings for your project.

Editor

Use this tab to configure settings for editor in Scala.

Item	Description
Show type info on mouse hover after, ms	Use this field to invoke Show Type Info action on a mouse hover after the specified time in the editor.
Highlighting	Use this area configure how your code is highlighted in the editor. You can select from the following options: – Highlight methods added via implicit conversion in code completion dialog- if you select this check box, IntelliJ IDEA highlights methods that are added via implicit conversions with a grey underline. – Highlight arguments to by-name parameters – Include block expressions – Include literals – Collection type highlighting - select this check box to highlight imports with prefixes.
Autocomplete	Use thia area to configure settings for the autocomplete action.
Ahead-of-time completion (parameter and variables names)	Select this check box to invoke autocompletion for both the name and the type before actually adding a type reference.
Use Scala classes priority over Java classes	Select this check box if you want IntelliJ IDEA to invoke the code completion for Scala classes before Java classes.
Code Conversion	Use this area to configure editor settings for converting Java code to Scala code automatically. The following options are available: – Convert Java code to Scala on copy-paste - select this check box to convert a code from Java files to Scala when you perform a copy-paste action. In this case IntelliJ IDEA displays Convert code from Java dialog. – Don't show on paste and automatically convert to Scala code - select this check box if you want to perform code conversion automatically without showing the Convert code from Java dialog.

Performance

Item	Description
Implicit parameters search depth (-1 for none)	
Search all symbols (include locals)	
Resolve to all classes, even in wrong directories (this may cause performance problems)	
Disable parsing of documentation comments. This may improve editor performance for very large files .(SCL-2900)	
Disable language injection in Scala files (injected language may freeze typing with auto pop-up completion)	
Don't cache compound types (use it in case of big pauses in GC)	
Use old implicit conversion search algorithm	

Worksheet

Item	Description
Output cutoff limit, lines	Use this field to set a line limit to the output of your worksheet's results.
Run worksheet in the compiler process	Select this check box if you want to run work sheet inside the Scala compile server.
Run worksheet in the interactive mode	Select this check box to display the code results automatically.
Use eclipse compatibility mode	In this case all statements inside the object are included in the worksheet's output.

Base packages

Use this tab to specify base packages.

Misc

Item	Description
Scala Test default super class	
Language injection settings for interpolated strings	

Updates

Item	Description
Plugin update channel	Use this field to update the Scala plugin. You can choose to download either *EAP*, *Release* or *Nightly* version. As soon as the specified version is available, IntelliJ IDEA will display an appropriate notification.

Scala Compile Server

File | Settings | Languages and Frameworks | Scala Compile Server

Use this page to configure an external compile server for your Scala project.

Item	Description
Use external compile server for Scala	Select this check box to activate options for Scala compile server.
JVM SDK	Use this drop-down list to select an SDK for the Scala compile server.
JVM maximum heap size, MB	Use this field to specify the maximum heap size available to the process that launches the compiler. The default 1024 Mb is suitable for most of the purposes.
JVM parameters	Use this field to type the string to be passed to the VM when IntelliJ IDEA launches the compiler. If you need more room to type, click to open the Java VM command line parameters dialog where the text entry area is larger.

SQL Dialects

File | Settings | Languages and Frameworks | SQL Dialects for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | SQL Dialects for OS X

This page lets you specify which SQL dialects are used in your project.

A dialect may be set at the project level, and also at the level of a directory or a file.

If a dialect is not specified explicitly, it is inherited from a higher hierarchical level.

The dialects specified explicitly are shown in black. The inherited values are shown in gray italic.

Note For this page to be available, the Database Tools and SQL plugin must be enabled. See Enabling and Disabling Plugins .

Item	Description
File/Directory	This column shows the hierarchy of files and folders in the project. Each row corresponds to a directory or a file.
SQL Dialect	Specify the SQL dialect to be used. Click the cell of interest and select the necessary option from the list. In addition to particular dialects, you can select: – Clear. This will clear the corresponding cell. As a result, an option from a higher hierarchical level will be inherited. – <Generic>. This means that no particular dialect is specified. As a result, basic SQL92-based coding assistance is provided including completion and highlighting for SQL keywords, and table and column names. Syntax error highlighting is not available. So the file contents are always shown as syntactically correct. Also, code reformatting isn't possible.

Play2

File | Settings | Languages and Frameworks | Play2

– Compiler

– Routes

– Other

Note This page is available when the Scala plugin is enabled. For more information, see Enabling and Disabling Plugins .

Compiler

Use this tab to manually specify settings for the Play2 compiler if they were not generated automatically during the project import or if the project was created with an external tool such as gen-idea.

Item	Description
Use SBT Watcher	Select this check box to separately execute an SBT process with `~compile:test` command. In this case IntelliJ IDEA monitors project sources and recompiles the project when changes are detected.
Use Play 2 compiler for this project	Select this check box to use the Play 2 compiler version to enable the Play 2 compiler for this project explicitly .
Don't compile the project within IDEA before run	Select this check box if you do not want to compile the project before run. SBT will additionally compile the necessary source files.
Play2 module	Use this field to specify the Play 2 module for this project.
Project uri	Use this field to specify the SBT project's root address. For the imported projects, the SBT project's root matches the content root of the IntelliJ IDEA project.
Additional SBT options	Use this field to specify additional options that you can pass directly to SBT. For example, `-Xmx2048M` .

Routes

Use this tab to configure settings for the routes file in your project.

Item	Description
Minimum space for routes formatting	Use this field to set a minimum space for routes formating.
Ignore URL depth in route files	Select this check box to ignore the URL depth in the routes files.
Re-format routes file on Enter	Select this check box to reformat new entry in your routes file when you press Enter.

Other

Use this tab to configure additional settings for your Pla2 project.

Item	Description
Exclude 'target' dir on refresh	By default, when you refresh a project the 'target' directory is excluded. Clear this check box if you want to include the 'target' directory when your refresh your project.
Set template imports manually	Select this check box to add the Play2 framework's imports manually to your project.

Template Data Languages

File | Settings | Languages and Frameworks | Template Data Languages for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | Template Data Languages for OS X

To get coding assistance for template data languages, specify where in your project those languages are used.

You can set a language for a file, directory or an entire project.

Item	Description
File/Directory	A tree representation of files and directories in your project.
Template data language	To set a language for an item (file, directory or the project), click the Template data language cell to the right of that item and select the language.

TypeScript

File | Settings | Languages & Frameworks | TypeScript for Windows and Linux

IntelliJ IDEA | Preferences | Languages & Frameworks | TypeScript for OS X

`Ctrl+Alt+S`

In this dialog, activate, deactivate, and configure the behaviour of the built-in TypeScript compiler that transpiles TypeScript code into JavaScript so the code can be executed in a browser and generates sources maps that ensure correspondence between the lines in the generated JavaScript code and the lines in the original TypeScript code.

Item	Description
Node interpreter	In this field, specify the location of the Node.js executable file. In most cases, IntelliJ IDEA detects the Node.js executable and fills in the field automatically.
TypeScript version	In this area, specify the version of the compiler to use. IntelliJ IDEA displays the default bundled version. To use a custom version of the compiler: 1. Download the `typescriptServices.js`, `lib.d.ts`, and `lib.es6.d.ts` files from https://github.com/Microsoft/TypeScript/ . 2. Click Edit in the Compiler version area. 3. In the Configure TypeScript Compiler dialog box that opens, choose Custom directory and specify the folder where the downloaded files are stored.
Use TypeScript Service	Select this check box to get native support from the TypeScript Language Service according to the up-to-date specifications. In this case syntax and error highlighting is performed based on the annotations retrieved from the TypeScript Language Service while code completion lists contain both suggestions from the TypeScript Language Service and suggestions calculated by IntelliJ IDEA itself.
Enable TypeScript Compiler	Select this check box to activate the built-in compiler. When the check box is selected, syntax highlighting and code completion are provided based only on the data from the built-in Typescript compiler. After you select this check box, the fields below on the page become active and you can configure the behaviour of the compiler.
Track changes	– When this check box is selected, the built-in compiler "wakes up" upon any change to a TypeScript file. – When this check box is cleared, the built-in compiler ignores changes to TypeScript files. To re-activate the compiler, open the TypeScript Compiler Tool Window (View | Tool Windows | TypeScript Compiler), and click the Compile All button on the toolbar. If you have not opened the TypeScript Compiler Tool Window yet and it is not available from the View menu, choose Help | Find Action, then find and launch the TypeScript Compile All action from the list.
Scope	From this drop-down list, choose the scope to apply the compiler in. The available options are: – Project Files: all the files within the project content roots (see Content Root and Configuring Content Roots). – Project Production Files: all the files within the project content roots excluding test sources. – Project Test Files: all the files within the project test source roots. – Open Files: all the files that are currently opened in the editor. VCS Scopes: these scopes are only available if your project is under version control . – Changed Files: all changed files, that is, all files associated with all existing changelists . – Default: all the files associated with the changelist `Default` . Alternatively, click the Browse button and configure a custom scope in the Scopes dialog box that opens. For more details on scopes, see the pages Scopes and Scopes dialog .
Set options manually	Choose this option to configure the behaviour of the built-in compiler manually: – In Command line options field, specify the command line options to be passed to the compiler. See the list of acceptable options at TSC arguments . Note that, the `-w` or `--watch` option (Watch input files) is irrelevant. – Select the Generate source maps check box to generate source maps that set correspondence between lines in your TypeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly. – Select the Compile main file only check box to have IntelliJ IDEA compile only a specific file and the files that are referenced from it and ignore all the other files in the project. This may be helpful if you have a dedicated `main.ts` file which only references other files. – Select the Use output path check box to have the built-in compiler store the generated JavaScript files and source map in a custom folder. Specify the path to this folder explicitly or use one of the listed available macros in the format: `$<macros_name>$` . The available macros are: – `FileDir` : the path to the folder where the file is stored. – `FileRelativeDir` : the path to the file directory relative to the project root. – `FileDirRelativeToProjectRoot` : the path to the file directory relative to the module content root the file belongs to. – `ModuleFileDir` : the path to the module root folder. – `SourcePath` : the path to the source folder under the content root to which the file belongs, see Configuring Content Roots and Configuring Folders Within a Content Root . – `FileDirRelativeToSourcePath` : the path to the file relative to the source folder under the content root to the file belongs.
Use tsconfig.json	Choose this option to have the built-in compiler analyze the code according to the settings specified in the `tsconfig.json` file .

TSLint

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools | TSLint for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools | TSLint for OS X

`Ctrl+Alt+S`

Use this page to activate integration with the TSLint code verifier and configure its behaviour and sensitivity.

Item	Description
Enable	Select this check box to have TSLint applied to verify the code in the current project. After that the other controls in the page are enabled.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the ... button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
TSLint Package	In this field, specify the location of the `tslint` package installed globally or in the current project, see Using TSLint Code Quality Tool .
Configuration File	In this area, appoint the configuration to use. By default, IntelliJ IDEA first looks for a `tslint.json` configuration file. IntelliJ IDEA starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. If no `tslint.json` file is found, TSLint uses its default embedded configuration file. Accordingly, you have to define the configuration to apply either in a `tslint.json` configuration file, or in a custom JSON configuration file, or rely on the default embedded configuration. – To have IntelliJ IDEA look for a `tslint.json` file, choose the Search for tslint.json option. If no `tslint.json` file is found, the default embedded configuration file will be used. – To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the ... button and select the relevant file from the dialog box that opens.
Additional Rules Directory	In this field, specify the location of the files with additional code verification rules. These rules will be applied after the rules from `tslint.json` or the above specified custom configuration file and accordingly will override them.

Web Contexts

File | Settings | Languages and Frameworks | Web Contexts

Use this dialog box to specify the Web context settings that IntelliJ IDEA uses to resolve Web paths in HTML and JSP for a file, directory, or the entire project.

The dialog box is available only when the Java Server Pages Integration plugin is installed and enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If it is not, enable it as described in Enabling and Disabling Plugins .

Item	Description
File/Directory	Select the file, or directory, or project to specify a Web context for. Tip Expand the nodes in the tree until you access the required item.
Web Context	In this drop-down list, select the relevant Web context. If you select the Clear option, Web paths in the corresponding file or directory are resolved using the Web context setting inherited from the parent.

XSLT

File | Settings | Languages and Frameworks | XSLT for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | XSLT for OS X

Ctrl+Alt+S

This page appears if XPath View+XSLT Support plugin is enabled. Use this page to configure XSLT support at the IntelliJ IDEA level.

Item	Description
Show Associated Files in Project View	Select this check box to have XML files associated with XSLT stylesheets displayed in the project view. Associated XML files are displayed below the corresponding stylesheets.

XSLT File Associations

File | Settings | Languages and Frameworks | XSLT File Associations for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | XSLT File Associations for OS X

Ctrl+Alt+S

Use this page to associate an XSLT stylesheet file with XML files. This is necessary to enable error highlighting and enhanced completion for element and attribute names in XSLT node-selections.

Item	Description
Project XSLT Files	The pane shows all the XSLT files in a project tree view, grouped by modules and their content roots.
Associated Files	The pane shows all the XML file associated with the selected XSLT file.
+	Click this button to select the XML file that should be associated with the selected XSLT file. See the description of the dialog that opens . The button is available only when an XSLT file is selected in the Project XSLT Files pane.
−	Click this button to cancel the association between the XML file selected in the Associated Files pane and the XSLT file selected in the Project XSLT Files pane.

PHP

File | Settings | Languages and Frameworks | PHP for Windows and Linux

for OS X

Ctrl+Alt+S

The page and all the pages under this node are available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Use this page to configure PHP development and unit testing support in the project by choosing one of the available PHP interpreters, see Configuring Local PHP Interpreters and Configuring Remote PHP Interpreters .

Item	Tooltip and Shortcut	Description
PHP language level		In this drop-down list, specify the PHP functionality scope to get coding assistance for. Each functionality scope is associated with PHP version that supports this functionality. No correlation between the PHP version used in the project and the language level is enforced. Although the language version of each interpreter is detected automatically, you can still tell IntelliJ IDEA to provide you with coding assistance that corresponds to another language level. However, if you attempt to use a code construct that is not supported by the specified language level, IntelliJ IDEA suggests a Switch to PHP <version> quick-fix .
Interpreter		From this drop-down list, choose the PHP interpreter to use in the current project. The list contains all the currently configured local and remote PHP interpreters. See Configuring Local PHP Interpreters and Configuring Remote PHP Interpreters for details.
	Reload	Click this button to make sure that the configuration you have chosen points at the relevant installation. If no PHP executable is detected at the specified directory, IntelliJ IDEA displays the corresponding error message.
	Show phpinfo	Click this button to have IntelliJ IDEA display a separate information window where you can examine the installation details and view the list of loaded extension and configured options. Please note that the options specified in the Configuration Options field of the Interpreters dialog box are not listed.
	Shift+Enter	Click this button next to the Interpreter drop-down list to create a new IntelliJ IDEA-wide PHP installation configuration in the Interpreters dialog box, that opens, see Configuring Local PHP Interpreters and Configuring Remote PHP Interpreters .
Path mappings		
Include path		The area shows a list of configured include paths. Include paths are used for code completion and reference resolution in some functions/methods that use file paths as arguments (for example, `require()` or `include()`). You also need to configure PEAR as an include path to run the PHPUnit test framework from it when the Load from include path option is chosen on the PHPUnit page of the Settings / Preferences Dialog . Use the Add + and Remove — buttons to manage the contents of the list. Use the Up and Down buttons to change the order of items in the list.

Servers

File | Settings | Languages and Frameworks | PHP | Servers for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Servers for OS X

Ctrl+Alt+S

The page is available only when the PHP plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

On this page, configure HTTP access for debugging engines to interact with local and remote Web servers and set correspondence between files on the server and their local copies in the IntelliJ IDEA project. The settings from debug server configurations are used when debugging with run/debug configurations of the type PHP Web Application or PHP Remote Debug and during Zero-Configuration Debugging sessions.

Toolbar and common options

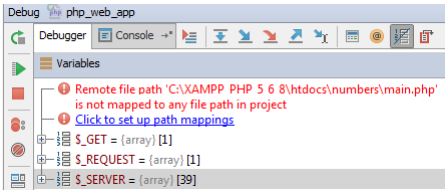
Use the toolbar buttons to manage the list of configurations.

Item	Tooltip and shortcut	Description
+	Add Insert	Click this button to define a new configuration.
−	Delete Delete	Click this button to remove the selected configuration from the list.
	Import	Click this button to open the Import from Deployment Configuration Dialog dialog box. In this dialog box, choose a configuration to access the application on the server and use the host and port settings from it. The dialog box also shows the path mappings retrieved from the deployment configuration. You need to transform relative paths on the server into absolute paths: – For an FTP, SFTP, or FTPS server access configuration, specify the absolute path to the server deployment root. This path will be added as a prefix to the path from the Root Path text box on the Deployment: Connection Tab . If you are not sure about this absolute path, you can open the Remote Host tool window, choose the required deployment configuration, position the cursor at the root folder, and choose Copy Path on the context menu, see Accessing Files on Web Servers for details. Alternatively, contact your hosting provider. – For a server access configuration of the type Local or Mounted Folder, specify the absolute path to the document root of the server or to the mounted folder. This path should be the one specified in the Folder field on the Deployment: Connection Tab . – For Inplace Server configurations no mappings are required because the local and remote paths are the same in this case. See Configuring Synchronization with a Web Server for details.

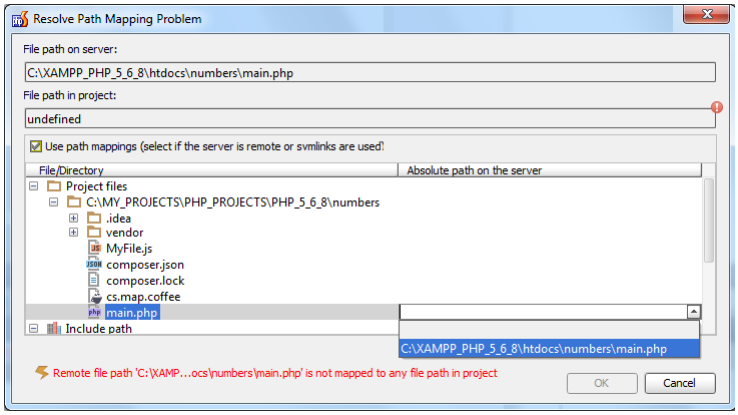
Configuration Details

In this area, specify the connection parameters and mappings to be used during debugging sessions. Note that the Validate Remote Environment button has been removed from this page, this functionality is now available through Run | Web Server Debug Validation on the main menu.

Item	Description
Name	In this text box, type the name of the server debug configuration.
Host	In this text box, type the name of the host where the target application is deployed.
Port	In this text box, type the port to connect to the specified host through. If you are using localhost on your machine, this setting should correspond with the port specified in the configuration file of the local Web server where the application will be executed or debugged.
Debugger	From this drop-down list, select the debug engine to use. The available options are: – Xdebug – Zend Debugger
Use path mappings	– Select this check box, if you are working on a remote Web server, that is, when the Web server is on a physically remote host, or the Web server is installed on your machine but your project is outside the Web server document root. Also select the check box if you are using symlinks. Map the absolute paths to the files and folders on the server with absolute paths to your project files in the local file system using the Path on server and File/Directory fields respectively. – File / Directory: This read-only field displays the files and folders of the current project. Select a file or a folder to be used as the local copy. – Path on server: In this filed, specify the absolute path to the file or folder on the target server to which the selected local file or folder corresponds. Type the path manually or select it from the drop-down list. – Clear this check box if you are working right on your Web server so your project root is under the server document root. In this case the absolute paths to the files on the Web server and the absolute paths to the corresponding files in your project are the same. If you do not specify any path mappings and start debugging an application that is not under the server document root, IntelliJ IDEA displays an error message:



The Click to set up path mappings link brings up the Resolve Path Mappings Problem dialog box, where you can define the path mappings:



When you click OK and leave the dialog box, IntelliJ IDEA selects the Use path mappings check box on the Servers page automatically.

Shared	Select this check box to share the debug server configuration across a team. The host/port settings and the path mappings are stored in the `.idea/php.xml` file is available to all team members through a version control system. Note that mappings are shared only for directories inside the project.

Import from Deployment Configuration Dialog

File | Settings | Languages and Frameworks | PHP | Servers for Windows and Linux

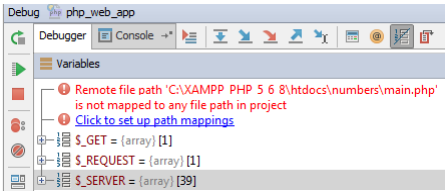
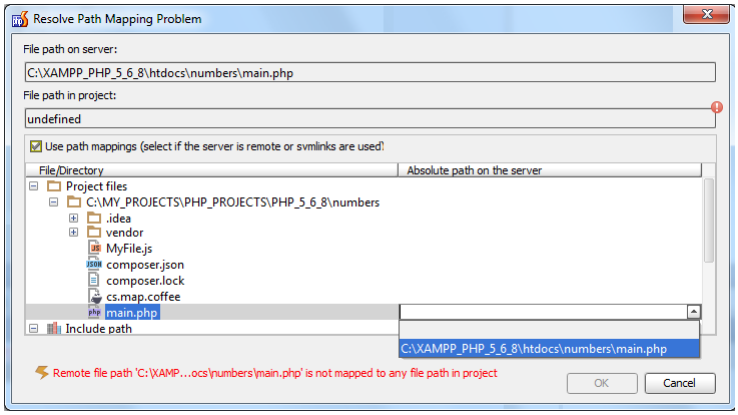
IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Servers for OS X

Ctrl+Alt+S

The dialog box is available only when the PHP and Remote Hosts Access plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

The dialog box opens when you click the Import button on the toolbar of the Servers page.

In this dialog box, choose a configuration to access the application on the server and use the host and port settings from it.

Item	Description
Deployment	From this drop-down list, choose the server access configuration (deployment configuration) to copy the server access settings from. The list contains all the available deployment configurations. To create a new configuration, click ... and specify new settings in the Deployment: Connection Tab dialog box that opens. In the text box below, specify the absolute path to the server deployment root folder, the name of the text box depends on the type of the selected server access configuration: Absolute path to the deployment root for FTP/ SFTP/ FTPS or Remote path to the mounted folder for Local or mounted folder. This path will be added as a prefix to the path from the Root Path text box on the Deployment: Connection Tab . If you are not sure about this absolute path, you can open the Remote Host tool window, choose the required deployment configuration, position the cursor at the root folder, and choose Copy Path on the context menu, see Accessing Files on Web Servers for details. Alternatively, contact your hosting provider.
Absolute path to the deployment root/ Remote path to the mounted folder	Depending on the type of the server access configuration chosen from the Deployment drop-down list, specify one of the following: – For an FTP, SFTP, or FTPS server access configuration, specify the absolute path to the server deployment root. This path will be added as a prefix to the path from the Root Path text box on the Deployment: Connection Tab . If you are not sure about this absolute path, you can open the Remote Host tool window, choose the required deployment configuration, position the cursor at the root folder, and choose Copy Path on the context menu, see Accessing Files on Web Servers for details. Alternatively, contact your hosting provider. – For Local or mounted folder, type the absolute path to the server root as specified in the Folder field of the Import from Deployment Configuration Dialog dialog box. – For Inplace Server configurations no mappings are required because the local and remote paths are the same in this case.
Preview	The area shows the host/port and the path mappings retrieved from the chosen server access configuration (deployment configuration). When you choose the deployment configuration to use, the Absolute path on the server field shows relative paths mapped to the project files and folders in the chosen configuration, that is, paths to files and folders relative to the deployment root. As you specify the absolute path to the deployment root (the server root for FTP/SFTP/FTPS or the mounted folder), the contents of the field are updated automatically and finally the field shows absolute paths on the server. Specify how IntelliJ IDEA will set up correspondence between files on the server and their local copies. Based on these mappings, IntelliJ IDEA will open local copies of currently processed files. Path mappings in PHP Debug Server configurations look very similar to the path mappings in server access (deployment configurations. Unfortunately, they cannot be reused, as deployment configurations uses relative paths while PHP Debug Servers configurations rely on absolute paths. – Select the Use path mappings check box if you are working on a remote Web server, that is, when the Web server is on a physically remote host, or the Web server is installed on your machine but your project is outside the Web server document root. Also select the check box if you are using symlinks. Map the absolute paths to the files and folders on the server with absolute paths to your project files in the local file system using the Path on server and File/Directory fields respectively. – File / Directory: This read-only field displays the files and folders of the current project. Select a file or a folder to be used as the local copy. – Path on server: In this filed, specify the absolute path to the file or folder on the target server to which the selected local file or folder corresponds. Type the path manually or select it from the drop-down list. – Clear the Use path mappings check box if you are working right on your Web server so your project root is under the server document root. In this case the absolute paths to the files on the Web server and the absolute paths to the corresponding files in your project are the same. If you do not specify any path mappings and start debugging an application that is not under the server document root, IntelliJ IDEA displays an error message:  The Click to set up path mappings link brings up the Resolve Path Mappings Problem dialog box, where you can define the path mappings:  When you click OK and leave the dialog box, IntelliJ IDEA selects the Use path mappings check box on the Servers page automatically.

Item **Description**

Composer

File | Settings | Languages and Frameworks | PHP | Composer for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Composer for OS X

Ctrl+Alt+S

The page is available only when the PHP and the Command Line Tool Support plugins are enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

On this page, enable the use of Composer Dependency Manager through the dedicated user interface by specifying the paths to the PHP engine installation and to the `composer.phar` archive to look for Composer commands in.

When the file analyses is completed, IntelliJ IDEA returns to the Command Line Tools page of the Settings dialog bos where the specified file is added to the list of command line tools available in IntelliJ IDEA.

Item	Description
Path to PHP executable	In this field, specify the location of the PHP engine installation folder. Choose the PHP installation from the list, or type the path manually, or click the Browse button … and choose the desired location in the dialog box that opens.
Path to composer.phar	In this text box, specify the location of the `composer.phar` archive. Type the path manually or click the Browse button … and choose the desired location in the dialog box that opens.
Click here to download from getcomposer.org	Click this link to have IntelliJ IDEA download `composer.phar` from the official storage and specify the folder to store the archive in. This instance of Composer will be available in the current project only. To use it in the command line mode, configure it as a command line tool .
Path to composer.json	In this text box, specify the location of the `composer.json` for listing descriptions of the project dependencies. Type the path manually or click the Browse button … and choose the desired location in the dialog box that opens.
Add packages as libraries	– Select this check box to have IntelliJ IDEA mark Composer packages added to the current project as libraries.. In the context of IntelliJ IDEA, any file marked as library is automatically write-protected. Library files are indexed for the only purpose which is enhancing coding assistance (i.e. code completion, syntax highlighting, navigation, and documentation lookup). Library files are not to be edited in the scope of the project where they are declared as library. By default, the Add packages as libraries check box is selected, which automatically blocks the possibility to edit them in the current project. – Clear the check box if you want to use Composer packages in your development and at the same time update them yourself within the scope of the same project.

Blade

File | Settings | Languages and Frameworks | PHP | Blade for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | Blade for OS X

`Ctrl+Alt+S`

This page is available only when the PHP and Blade plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

The page consists of the following tabs:

- Text Tags
- Directives

Text Tags

In this tab, specify the delimiters to use in Blade templates. According to these settings, IntelliJ IDEA recognizes templates and provides error highlighting and code completion for them.

The fields in the tab show the opening and closing characters for raw tags , content tags , and escaped tags .

The fields are filled in with the default values in compliance with Blade Templates 5.1 . If you are using an earlier version, you can specify the relevant custom delimiters and IntelliJ IDEA will provide coding assistance according to the new rules.

Directives

In this tab, manage Blade directives for use in IntelliJ IDEA. The tab lists all the currently available Blade directives, for those that have parameters, the prefixes and suffixes are also shown. When you start, the list contains only predefined directives. You can edit these directives as well as create custom ones.

Item	Description
Directives	The tab lists all the currently available Blade directives, for those that have parameters, the prefixes and suffixes are also shown. When you start, the list contains only predefined directives. You can edit these directives as well as create custom ones. – Add directive (+) click this button to define a new directive. Specify the directive's name in the Name text box. If the new directives requires a prefix and a suffix, select the Has parameter check box and type the prefix and suffix to use in the Prefix and Suffix text boxes respectively. IntelliJ IDEA will automatically enclose the prefix and suffix in opening and closing brackets and quotes and add a colon separator `:` so the parameters will look as follows: `("<prefix>:<suffix>")` . To edit an existing directive, select it in the list and change the values in the text boxes below. To restore the original definition, click the Reset to defaults button . – Remove directive(s) (—) click this button to remove the selected directive from the list. – Reset to defaults ( ) click this button to restore the original definition of the selected directive.
Name	In this text box, specify the name of a new directive or edit the name of the selected one.
Has parameter	Select this check bo to specify a prefix and a suffix for a new directive or for the one selected in the list. When the check box is selected, the Prefix and Suffix text boxes are available for editing. Specify the required directive parameters. If the new directives requires a prefix and a suffix, select the Has parameter check box and type the prefix and suffix to use in the Prefix and Suffix text boxes respectively. IntelliJ IDEA will automatically enclose the prefix and suffix in opening and closing brackets and quotes and add a colon separator `:` so the parameters will look as follows: `("<prefix>:<suffix>")` .

Mess Detector

On this page:

- Mess Detector Page
- Mess Detector Dialog

Mess Detector Page

File | Settings | Languages and Frameworks | PHP | Mess Detector for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | Mess Detector for OS X

The page is available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

On this page, choose the Mess Detector script to use.

Item	Description
Configuration	From this drop-down list, choose the script to use: – To use the script associated with a specific remote PHP interpreter, choose the name of this interpreter. – To use the script associated with the default project interpreter, that is, the one chosen on the PHP page of the Settings dialog box, choose By default project interpreter. – To use a local script, choose Local. In this case the local Mess Detector will be executed no matter which PHP interpreter - local or remote - is used in the project. Note that there can be only one Local configuration for Mess Detector because IntelliJ IDEA runs a script (`phpmd.bat` for Windows or `phpmd` for Linux) which contains a path to a PHP engine.
Ignored files	This area displays a list of files that Mess Detector skips. IntelliJ IDEA suggests adding a new file to the list during inspection when waiting for response from the Mess Detector exceeds the limit specified in the Tool process timeout field. This is done to prevent slowing down processing. For each file, IntelliJ IDEA displays its name and location. – To delete a file from the list and have Mess Detector process it again, select the file and click the Remove file button . – To remove all the files from the list, click the Clean the list button .

Mess Detector Dialog

The dialog box opens when you click … next to the Configuration drop-down list on the Mess Detector page.

Use this dialog box to configure local Mess Detector scripts, scripts associated with remote PHP interpreters, see Using PHP Mess Detector , and configure Mess Detector's behaviour.

The left-hand pane of the dialog box shows all the configured Mess Detector scripts, one of them is of the type Local, and others are named after the remote PHP interpreters with which the scripts are associated. When you select a configuration, the right-hand pane shows its details.

– To configure or edit the Local script, select Local and specify the location of `phpmd.bat` or `phpmd` in the PHP Mess Detector path field.

– To configure a new script associated with a remote PHP interpreter:

1. Click + on the toolbar.
2. In the Mess Detector by Remote Interpreter dialog box that opens, choose the remote PHP interpreter to use the associated script from. If the list does not contain a relevant interpreter, click … and configure a remote interpreter in the Interpreters dialog box as described in Configuring Remote PHP Interpreters .
   When you click OK, IntelliJ IDEA brings you back to the Mess Detector dialog box where the new Mess Detector configuration is added to the list and the right-hand pane shows the chosen remote PHP interpreter, the path to the Mess Detector associated with it, and the advanced PHP Mess Detector options.

Item	Description
PHP Mess Detector (phpmd) Path	In this text box, specify the location of the Mess Detector utility `phpmd` or `phpmd.bat` .If the script is associated with a PHP interpreter, IntelliJ IDEA detects the path to it and fills in the field automatically but you can edit it if necessary. In either case, type the path manually or click the Browse button … and select the path in the dialog box, that opens. To check that the specified path to `phpmd.bat` or `phpmd` ensures interaction between IntelliJ IDEA and Mess Detector, that is, the tool can be launched from IntelliJ IDEA and IntelliJ IDEA will receive problem reports from it, click the Validate button. This validation is equal to running the `phpmd --version` command. If validation passes successfully, IntelliJ IDEA displays the information on the detected Mess Detector version.
Interpreter	The field shows the chosen PHP interpreter to use the Mess Detector from.
Maximum number of messages per file	In this text box, set the upper limit for the total number of messages to be reported for a file. All the messages above this limit will be rejected. IntelliJ IDEA will display the following warning right in the code: `Too many PHP Mess Detector messages` and suggest adding the file to the Ignored files list.
Tool process timeout	In this text box, specify how long you want IntelliJ IDEA to wait for a result from PHP Mess Detector, whereupon the process is killed to prevent excessive CPU and memory usage. This gives you the capability to fine tune the PHP Mess Detector process behavior depending on the configuration of your computer and the rule sets used.

Framework Integration

File | Settings | Languages and Frameworks | PHP | Framework Integration for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Framework Integration for OS X

Ctrl+Alt+S

The page is available only when the PHP and the Framework MVC Structure Support plugins are enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Use this page to enable viewing the structure of your framework-specific project in the terms of the Model-View-Controller pattern .

IntelliJ IDEA parses the structure of a framework-specific project tree based on the following assumptions:

1. According to the framework specification, a project should have a directory with application configuration files, a `web` directory with `js` and `css` files, and directory with a number of modules. The structure of the application `config` file and `web` directories is predefined and cannot be customized, therefore IntelliJ IDEA can parse such directories itself, you do not need to provide any additional information.
2. However inside the project, the directories can be arranged in any way and can be named without any restrictions. Moreover, each module should contain some specific folders that can be located anywhere inside the module and can have any name. Therefore, IntelliJ IDEA cannot detect the type of such folders itself.

To enable the MVC view of the project, you need to mark application roots and module roots and mark the project elements as various framework-specific types.

Item	Description
Framework name	In this drop-down list, specify the framework your project is based on. The available options are: – Symfony2 – Yii
File/Directory	In this field, select the file or folder to associate with a framework-specific type.
Element	In this field, specify the framework-specific type of element for the selected file or folder. The available options for Symfony2 are: – Resources – app for directories with application settings – src – vendor – web for web roots – Bundle root: choose this option to mark a directory as the root of a module or a Symfony bundle. Modules/bundles usually have their own types inside, so subdirectories under a bundle root can be marked as: – Controller – Dependency/Injection – Form – Tests The available options for Yii are: – <none> – Application base directory – Assets – Controllers – Extensions – Layouts – Models – Runtime directory – System view directory – Themes – Views – Module root : – Components – Controllers – Extensions – Messages – Models – Views

Behat

File | Settings | Languages and Frameworks | PHP | Behat for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Behat for OS X

Ctrl+Alt+S

This page is available only when the PHP and Behat plugins are installed and enabled. The plugins are not bundled with IntelliJ IDEA, but they can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Once enabled, the plugins are available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects.

Before you start, make sure the PHP interpreter is configured in IntelliJ IDEA on the PHP page , as described in Enabling PHP Support .

On this page, configure the Behat framework to enable behaviour-driven development in IntelliJ IDEA.

Item	Description
Interpreter	In this field, IntelliJ IDEA displays the appointed PHP interpreter configuration to use with Behat. The field is read-only, to update the chosen interpreter configuration or create a new one, click the ... whereupon IntelliJ IDEA brings you to the Interpreters dialog box .
Behat Library	In this area, specify the Behat installation to use. – Path to Behat directory of phar file: in this text box, specify the location of the `behat.phar` archive or the folder with the Behat executable file. Behat does not necessarily have to be installed under the current project root. You can type the path manually or click the ... button and choose the relevant location in the dialog box that opens. – Download behat.phar from https://github.com/downloads/Behat/Behat/behat.phar: click this link to have IntelliJ IDEA download the `behat.phar` archive. This approach works only if you have already an account on GitHub and you have configured access to this account in IntelliJ IDEA, see Registering GitHub Account in IntelliJ IDEA . – Behat version: this read-only field shows the version of the specified Behat installation. IntelliJ IDEA detects the version when you click the Refresh icon . The default value is Not installed
Test Runner	In this area, appoint the configuration `.yaml` file to use for launching and executing scenarios. By default, Behat looks for a `behat.yaml` configuration file in the project root folder or in the `config` folder. You can appoint a custom configuration file. – Default configuration file: select this check box to specify your own `.yaml` configuration file. Ths file will be later used as default in all Behat run/debug configurations. In the text box, specify the location of the configuration file to use. Type the path manually or click the ... button and choose the file in the dialog box that opens. Clear the check box to have Behat use the `behat.yaml` configuration file from the project root folder or from the `config` folder. If no such file is found, test execution fails, therefore it may be more reliable to specify the configuration file explicitly.

Debug

File | Settings | Languages and Frameworks | PHP | Debug for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Debug for OS X

Ctrl+Alt+S

The page is available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Use this page to configure the behaviour of the Xdebug and Zend Debugger debugging tools.

Item	Description
Evaluation	– Show array and object children in Debug Console: Select this check box to show the output for arrays and objects in the Console pane, see Debug Tool Window. Console . When the check box is cleared, the output is not displayed. – Safe evaluation mode in value hints and Watches Frame: – When this check box is selected, IntelliJ IDEA checks that the expression or code fragment to be evaluated does not contain any undefined elements and informs you about any discrepancies detected. – If the check box is cleared, an exception appears if IntelliJ IDEA encounters any undefined elements during evaluation. See Evaluating Expressions for details.
Local debug	In this area, customize local debugging, when the processed files are on your machine. – Pass required configuration options through command line (still need to enable debug extension manually): select this check box to have debugger configuration options passed through a command line.
External Connections	In this area, specify how you want IntelliJ IDEA to treat connections received from hosts and through ports that are not registered as server configurations . – Ignore external connections through unregistered server configurations: Select this check box to have IntelliJ IDEA ignore connections received from hosts and through ports that are not registered as server configurations . When this check box is selected, IntelliJ IDEA does not attempt to create a server configuration automatically. – Detect path mappings from deployment configurations: – When this check box is selected, IntelliJ IDEA attempts to retrieve path mappings for debugging in a remote environment from the server access configuration (deployment configuration). – When the check box is cleared, you have to specify the path mappings manually. See Validating the Configuration of a Debugging Engine , Web Server Debug Validation Dialog , and Configuring Remote PHP Interpreters for details. – Break at first line in PHP scripts: Select this check box to have the debugger stop as soon as connection between it and IntelliJ IDEA is established (instead of running automatically until the first breakpoint is reached). Alternatively turn on the Run | Break at first line in PHP scripts option on the main menu. – Max. simultaneous connections: Use this spin box to limit the number of external connections that can be processed simultaneously.
Xdebug	Use the controls in this area to configure debugging using the Xdebug tool. – Debug port: in this text box, specify the port for IntelliJ IDEA and the Xdebug engine to communicate through. This must be exactly the same port number as specified in the `php.ini` file: `xdebug.remote_port = <port_number>` By default, Xdebug listens on port 9000. – Can accept external connections: select this check box to enable IntelliJ IDEA to accept any incoming connections from Xdebug engines through the port specified in the Debug port text box. – Force break at the first line when no path mapping is specified: Select this check box to have the debugger stop as soon as it reaches and opens a file that is not mapped to any file in the project on the Servers page. The debugger stops at the first line of this file and Debug Tool Window. Variables shows the following error message: Cannot find a local copy of the file on server <path to the file on the server> and a link Click to set up mappings. Click the link to open the Resolve Path Mappings Problem dialog box and map the problem file to its local copy. When this check box cleared, the debugger does not stop upon reaching and opening an unmapped file, the file is just processed, and no error messages are displayed. – Force break at the first line when the script is outside the project: Select this check box to have the debugger stop at the first line as soon as it reaches and opens a file outside the current project. With this check box cleared, the debugger continues upon opening a file outside the current project.
Zend Debugger	Use the controls in this area to configure debugging using the Zend Debugger tool. – Debug port: In this text box, specify the port for IntelliJ IDEA and the Zend Debugger engine to communicate through. Type the port number within the tunnel specified in the `php.ini` file through `zend_debugger.tunnel_min_port` and `zend_debugger.tunnel_max_port` . For details, see http://files.zend.com/help/previous-version/Zend-Server-4-Community-Edition/zenddebugger.html – Can accept external connections: Select this check box to enable IntelliJ IDEA to accept any incoming connections from Zend Debugger engines through the port specified in the Debug port text box. – Settings broadcasting port: In this text box, specify the port through which the debugger settings are passed to the debugging toolbar in the browser. – Ignore Z-Ray system requests: Select this check box to block requests from the Z-Ray system if they annoy you by invoking the IntelliJ IDEA debugger too often. Tip If starting the Zend Debugger tool fails with the message "Port is busy", specify a port number of your choice higher than 10000.
Use debugger bookmarklets to initiate debugger from your favorite browser	Follow this link to open the Zend Debugger & Xdebug bookmarklets page where you can generate bookmarklets through which you will start/stop a debugging session by controlling the debugger cookie .
Advanced Settings	Notify if debug session was finished without being stopped: Select this check box to have IntelliJ IDEA display a notification when no breakpoints are hit during Zero-Configuration debugging. This may happen if the path mappings are not configured or configured erroneously, or if you have not set any breakpoints. In the latter case, enabling selecting the Break at First Line in PHP Scripts check box in the External Connections area or turning the Run | Break at First Line in PHP Scripts option on the main menu may also help. If the check box is cleared, no debugging sessions is established and the PHP script is just executed without being suspended.

Skipped Paths

File | Settings | Languages and Frameworks | PHP | Debug | Skipped Paths for Windows and Linux

for OS X

Ctrl+Alt+S

The page is available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

On this page, specify the scripts requests to which you want IntelliJ IDEA to ignore during debugging. This approach can be useful, when your application contains scripts that use AJAX. Suppose you have a `menu-ajax-script.php` that "reloads" a part of your web page. This script works properly so you do not need to debug it. However, this script is still requested during the debugging session. To have incoming connections to this script ignored, add the `menu-ajax-script.php` script to the skipped paths list. You can also group such scripts into folders and add these folders to the "ignore list".

Item	Description
Notify about skipped paths	Select this check box to have IntelliJ IDEA inform you every time it receives a request to a script to be skipped.
Skipped paths	This list box shows the scripts and folders to ignore requests to.
+ Add (Alt+Insert)	Click this button to have a new line added to the list. Then click the Browse button ... and in the dialog box that opens choose the file or folder to skip connections to.
– Remove (Alt+Delete)	Click this button to have the selected item removed from the list.

DBGp Proxy

File | Settings | Languages and Frameworks | PHP | Debug | DBGp Proxy for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Debug | DBGp Proxy for OS X

Ctrl+Alt+S

The page is available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

On this page, enable, disable, and re-configure your access to debugging PHP applications in the multiuser mode via an Xdebug proxy server .

Item	Description
IDE key	In this text box, specify the name for the Proxy server to identify connections from your IDE. This should be the value of the `xdebug.idekey` setting in your currently active `php.ini` configuration file.
Host	In this text box, specify the host on which the Xdebug proxy server resides.
Port	In this text box, specify the port which IntelliJ IDEA will listen to during a proxy debugging session.

Code Sniffer

On this page:

- Code Sniffer Page
- Code Sniffer Dialog

Code Sniffer Page

File | Settings | Languages and Frameworks | PHP | Code Sniffer for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Code Sniffer for OS X

`Ctrl+Alt+S`

The page is available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

On this page, choose the Code Sniffer script to use.

Item	Description
Configuration	From this drop-down list, choose the script to use: – To use the script associated with a specific remote PHP interpreter, choose the name of this interpreter. – To use the script associated with the default project interpreter, that is, the one chosen on the PHP page of the Settings dialog box, choose By default project interpreter. – To use a local script, choose Local. In this case the local Code Sniffer will be executed no matter which PHP interpreter - local or remote - is used in the project. Note that there can be only one Local configuration for Code Sniffer because IntelliJ IDEA runs a script (`phpcs.bat` for Windows or `phpcs` for Linux) which contains a path to a PHP engine.
Ignored files	This area displays a list of files that Code Sniffer skips. IntelliJ IDEA suggests adding a new file to the list during inspection when waiting for response from the Code Sniffer exceeds the limit specified in the Tool process timeout field. This is done to prevent slowing down processing. For each file, IntelliJ IDEA displays its name and location. – To delete a file from the list and have Code Sniffer process it again, select the file and click the Remove file button —. – To remove all the files from the list, click the Clean the list button ✖.

Code Sniffer Dialog

The dialog box opens when you click … next to the Configuration drop-down list on the Code Sniffer page.

Use this dialog box to configure local Code Sniffer scripts, scripts associated with remote PHP interpreters, see Using PHP Code Sniffer Tool , and configure Code Sniffer's behaviour.

The left-hand pane of the dialog box shows all the configured Code Sniffer scripts, one of them is of the type Local, and others are named after the remote PHP interpreters with which the scripts are associated. When you select a configuration, the right-hand pane shows its details.

– To configure or edit the Local script, select Local and specify the location of `phpcs.bat` or `phpcs` in the PHP Code Sniffer path field.

– To configure a new script associated with a remote PHP interpreter:

1. Click + on the toolbar.
2. In the Code Sniffer by Remote Interpreter dialog box that opens, choose the remote PHP interpreter to use the associated script from. If the list does not contain a relevant interpreter, click … and configure a remote interpreter in the Interpreters dialog box as described in Configuring Remote PHP Interpreters .
   When you click OK, IntelliJ IDEA brings you back to the Code Sniffer dialog box where the new Code Sniffer configuration is added to the list and the right-hand pane shows the chosen remote PHP interpreter, the path to the Code Sniffer associated with it, and the advanced PHP Code Sniffer options.

Item	Description
PHP Code Sniffer (phpcs) Path	In this text box, specify the location of the Code Sniffer utility `phpcs` or `phpcs.bat` .If the script is associated with a PHP interpreter, IntelliJ IDEA detects the path to it and fills in the field automatically but you can edit it if necessary. In either case, type the path manually or click the Browse button … and select the path in the dialog box, that opens. To check that the specified path to `phpcs.bat` or `phpcs` ensures interaction between IntelliJ IDEA and Code Sniffer, that is, the tool can be launched from IntelliJ IDEA and IntelliJ IDEA will receive problem reports from it, click the Validate button. This validation is equal to running the `phpcs --version` command. If validation passes successfully, IntelliJ IDEA displays the information on the detected Code Sniffer version.
Interpreter	The field shows the chosen PHP interpreter to use the Code Sniffer from.
Maximum number of messages per file	In this text box, set the upper limit for the total number of messages to be reported for a file. All the messages above this limit will be rejected. IntelliJ IDEA will display the following warning right in the code: `Too many PHP Code Sniffer messages` and suggest adding the file to the Ignored files list.
Tool process timeout	In this text box, specify how long you want IntelliJ IDEA to wait for a result from PHP Code Sniffer, whereupon the process is killed to prevent excessive CPU and memory usage. This gives you the capability to fine tune the PHP Code Sniffer process behavior depending on the configuration of your computer and the rule sets used.

Google App Engine for PHP

File | Settings | Languages&Frameworks | Node.js and NPM for Windows and Linux

IntelliJ IDEA | Preferences | Languages&Frameworks | Node.js and NPM for OS X

Ctrl+Alt+S

In this dialog box, activate and configure the support of the Google App Engine for PHP in an existing IntelliJ IDEA project. For more details, see Preparing to Develop a Google App for PHP Application .

Item	Description
Enable Google App Engine for PHP support	– When this check box is selected, IntelliJ IDEA provides assistance in developing PHP applications intended for running in the Google PHP Runtime Environment . This support includes coding assistance and the possibility to run and debug such PHP applications on the local Google development server. – When the check box is cleared, no assistance in developing PHP applications for running in the Google runtime environment is provided and all the controls on the page are disabled.
SDK directory	In this field, specify the folder where the Google App Engine for PHP SDK is stored.
Python executable	In this field, specify the path to the Python executable file, version 2.7 is required.
App Engine Account Settings	In this area, choose the way to authenticate to the development server, the available options are: – Use passwordless login via OAuth2: choose this option to use the OAuth 2.0 protocol . To save the token achieved through the Google Developers Console , clear the Do not save token check box. – Log in with email and password: choose this option to use your Gmail address and password.

Smarty

File | Settings | Languages and Frameworks | PHP | Smarty for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | Smarty for OS X

Ctrl+Alt+S

The page is available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Use this dialog box to specify the delimiters to enclose Smarty tags in.

Item	Description
Smarty left delimiter	In this text box, specify the desired opening delimiter. By default, the field shows the opening curly brace `{` .
Smarty right delimiter	In this text box, specify the desired closing delimiter. By default, the field shows the closing curly brace `}` .
Use Smarty 3 whitespace policy	When this check box is selected, delimiters followed by whitespace will not be parsed as template tags. The check box is available when Smarty 3.0 or higher is used.

PHPUnit

File | Settings | Languages and Frameworks | PHP | PHPUnit for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP | PHPUnit for OS X

The page is available only when the PHP plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Use this page to configure the way to launch unit tests of your PHP applications.

Item	Description
PHPUnit Library	In this area, specify where to take the PHPUnit test framework. The available options are: – Load from include path: choose this option to have IntelliJ IDEA run the PHPUnit from PEAR configured as an include path on the PHP page of the Settings / Preferences Dialog . – Use custom loader: choose this option to run the PHPUnit instance installed by the Composer Dependency Manager . The package is retrieved and loaded by the dedicated `autoload.php` file located in the `Vendor` folder. Specify the location of the `autoload.php` file in the Path to script text box. Type the path manually or click the Browse button … and select the file in the dialog box that opens, – Path to phpunit.phar: choose this option to run PHPUnit from the `.phar` archive. – If you already have the `phpunit.phar` archive on your computer, specify its location in the Path to phpunit.phar text box. Type the path manually or click the Browse button … and select the file in the dialog box that opens, – If you have no `.phar` archive on your computer yet, click the Download phpunit.phar ... link to have IntelliJ IDEA download it automatically. In either case, IntelliJ IDEA will load the archive before test execution.
Test Runner	In this area, appoint the configuration `XML` file to use for launching and executing scenarios. By default, PHPUnit looks for a `phpunit.xml` configuration file in the project root folder or in the `config` folder. You can appoint a custom configuration file. – Default configuration file: select this check box to specify your own `XML` configuration file. Ths file will be later used as default in all PHPUnit run/debug configurations. In the text box, specify the location of the configuration file to use. Type the path manually or click the … button and choose the file in the dialog box that opens. Clear the check box to have PHPUnit use the `phpunit.xml` configuration file from the project root folder or from the `config` folder. If no such file is found, test execution fails, therefore it may be more reliable to specify the configuration file explicitly. – Default bootstrap file: select this check box to have PHP script always executed before launching tests. In the text box, specify the location of the script. Type the path manually or click the Browse button … and select the desired folder in the dialog that opens .

Interpreters

File | Settings | Languages and Frameworks | PHP for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP for OS X

`Ctrl+Alt+S`

The dialog box opens when you click the Browse button … next to the Interpreter drop-down list in the Development environment section of the PHP page.

Use this dialog box to configure PHP engines as interpreters, see Configuring Local PHP Interpreters and Configuring Remote PHP Interpreters . In this dialog box, you can add new interpreters and edit or remove the existing ones.

The dialog box consists of two panes. The left-hand pane shows a list of all configured PHP interpreters, both local and remote ones, and a toolbar for adding, removing, and copying PHP interpreter configurations. The contents of the right-hand pane depend on the type of the interpreter selected in the left-hand pane. Use the control in this pane to edit the settings of the selected interpreter and even choose its type, if necessary.

- Left-hand pane
- Right-hand pane
- Configuration directives dialog

Left-hand pane

The left-hand pane shows a list of all configured PHP interpreters, both local and remote ones, and a toolbar for adding, removing, and copying PHP interpreter configurations.

Item	Tooltip	Description
+	Add	Click this button to add a new PHP interpreter to the list. From the drop-down list, choose the type of the interpreter (Local or Remote), and configure a local interpreter in the right-hand pane or a remote interpreter in the Configure Remote PHP Interpreter dialog that opens.
−	Delete	Click this button to remove the selected interpreter from the list.
	Copy	Click this button to create a new interpreter with the settings copied from the selected one.

Right-hand pane

The contents of the right-hand pane depend on the type of the interpreter selected in the left-hand pane. Use the control in this pane to edit the settings of the selected interpreter and even choose its type, if necessary.

Type of the selected interpreter	Description
Local	– Name: In this text box, type the identifier to distinguish the interpreter from others, for example, `php_installation_<version>` . – PHP executable: In this text box, specify the path to the PHP engine. Type the path manually or click the Browse button … and choose the executable file in the dialog box that opens. IntelliJ IDEA detects the version of PHP and displays it in the PHP version read-only field. IntelliJ IDEA also detects the debugging engine and the `php.ini` configuration file. The type of the debugging engine associated with the PHP interpreter and its version are displayed in the Debugger read-only field. If no debugger is detected or you have disabled it in `php.ini` file (see Configuring Xdebug for Using in the On-Demand Mode), the field shows Debugger: Not installed. The location of `php.ini` is displayed in the Configuration file read-only field. To edit the `php.ini` in IntelliJ IDEA, click Open in Editor. – (Reload): Click this button to check that the specified PHP home directory actually contains a PHP executable file. If no PHP executable is detected at the specified location, IntelliJ IDEA displays the corresponding error message. – (Show phpinfo): Click this button to have IntelliJ IDEA display a separate information window where you can examine the installation details and view the list of loaded extension and configured options. Please note that the options specified in the Configuration Options field of the Interpreters dialog box are not listed. – Debugger extension: in this text box, specify the location of the Xdebug extension to enable IntelliJ IDEA to activate it when necessary. Starting with version 2016.2, IntelliJ IDEA supports the On-Demand mode where you can disable debugger for your global PHP installation, and have it enabled automatically on demand only when you are debugging your command-line scripts or when you need code coverage reports. This lets your command line scripts (including Composer and unit tests) run much faster. – Configuration Options: Use this text box to customize the configuration settings of the installation by composing a string of configuration directives to be passed through the -d command line option and thus add new entries to the `php.ini` file. Click the Browse button … to open the Configuration Directives dialog box and create a list of new `php.ini` entries there.

Remote interpreter accessible through SSH	– Host: in this field, type the name of the host on which the interpreter is installed.
	– Port: in this field, type the port which the SSH server on the remote host listens to. The default port number is 22.
	– User name: in the field, type the user name under which you are registered on the SSH server.
	– Auth type: from this drop-down list, choose the authentication method.
	  – To access the host through a password, choose Password from the Auth type drop-down list and type the password.
	  – To access the host through a pair of SSH keys, choose Key pair, then specify the path to the file where your private key is stored or the passphrase if you have configured it during the generation of the key pair.
	– Name: In this text box, type the identifier to distinguish the interpreter from others, for example, `php_installation_<version>` .
	– PHP executable: In this text box, specify the path to the PHP engine. Type the path manually or click the Browse button … and choose the executable file in the dialog box that opens. IntelliJ IDEA detects the version of PHP and displays it in the PHP version read-only field.
	IntelliJ IDEA also detects the debugging engine and the `php.ini` configuration file. The type of the debugging engine associated with the PHP interpreter and its version are displayed in the Debugger read-only field. If no debugger is detected or you have disabled it in `php.ini` file (see Configuring Xdebug for Using in the On-Demand Mode), the field shows Debugger: Not installed.
	
	The location of `php.ini` is displayed in the Configuration file read-only field. To edit the `php.ini` in IntelliJ IDEA, click Open in Editor.
	– (Reload): Click this button to check that the specified PHP home directory actually contains a PHP executable file. If no PHP executable is detected at the specified location, IntelliJ IDEA displays the corresponding error message.

	– (Show phpinfo): Click this button to have IntelliJ IDEA display a separate information window where you can examine the installation details and view the list of loaded extension and configured options. Please note that the options specified in the Configuration Options field of the Interpreters dialog box are not listed. – Debugger extension: in this text box, specify the location of the Xdebug extension to enable IntelliJ IDEA to activate it when necessary. Starting with version 2016.2, IntelliJ IDEA supports the On-Demand mode where you can disable debugger for your global PHP installation, and have it enabled automatically on demand only when you are debugging your command-line scripts or when you need code coverage reports. This lets your command line scripts (including Composer and unit tests) run much faster. – Configuration Options: Use this text box to customize the configuration settings of the installation by composing a string of configuration directives to be passed through the -d command line option and thus add new entries to the `php.ini` file. Click the Browse button to open the Configuration Directives dialog box and create a list of new `php.ini` entries there.
Remote interpreter on a Vagrant instance	– Name: In this text box, type the identifier to distinguish the interpreter from others, for example, `php_installation_<version>` . – PHP executable: In this text box, specify the path to the PHP engine. Type the path manually or click the Browse button and choose the executable file in the dialog box that opens. IntelliJ IDEA detects the version of PHP and displays it in the PHP version read-only field. IntelliJ IDEA also detects the debugging engine and the `php.ini` configuration file. The type of the debugging engine associated with the PHP interpreter and its version are displayed in the Debugger read-only field. If no debugger is detected or you have disabled it in `php.ini` file (see Configuring Xdebug for Using in the On-Demand Mode), the field shows Debugger: Not installed. The location of `php.ini` is displayed in the Configuration file read-only field. To edit the `php.ini` in IntelliJ IDEA, click Open in Editor. – (Reload): Click this button to check that the specified PHP home directory actually contains a PHP executable file. If no PHP executable is detected at the specified location, IntelliJ IDEA displays the corresponding error message. – (Show phpinfo): Click this button to have IntelliJ IDEA display a separate information window where you can examine the installation details and view the list of loaded extension and configured options. Please note that the options specified in the Configuration Options field of the Interpreters dialog box are not listed. – Debugger extension: in this text box, specify the location of the Xdebug extension to enable IntelliJ IDEA to activate it when necessary. Starting with version 2016.2, IntelliJ IDEA supports the On-Demand mode where you can disable debugger for your global PHP installation, and have it enabled automatically on demand only when you are debugging your command-line scripts or when you need code coverage reports. This lets your command line scripts (including Composer and unit tests) run much faster. – Configuration Options: Use this text box to customize the configuration settings of the installation by composing a string of configuration directives to be passed through the -d command line option and thus add new entries to the `php.ini` file. Click the Browse button to open the Configuration Directives dialog box and create a list of new `php.ini` entries there.
Remote interpreter accessible through a deployment configuration	– Deployment Configuration: from this drop-down list, choose the server access configuration of the type SFTP according to which you want IntelliJ IDEA to connect to the target host. If the settings specified in the chosen configuration ensure successful connection, IntelliJ IDEA displays the URL address of the target host as a link in the Deployment Host URL field. – Name: In this text box, type the identifier to distinguish the interpreter from others, for example, `php_installation_<version>` . – PHP executable: In this text box, specify the path to the PHP engine. Type the path manually or click the Browse button and choose the executable file in the dialog box that opens. IntelliJ IDEA detects the version of PHP and displays it in the PHP version read-only field. IntelliJ IDEA also detects the debugging engine and the `php.ini` configuration file. The type of the debugging engine associated with the PHP interpreter and its version are displayed in the Debugger read-only field. If no debugger is detected or you have disabled it in `php.ini` file (see Configuring Xdebug for Using in the On-Demand Mode), the field shows Debugger: Not installed. The location of `php.ini` is displayed in the Configuration file read-only field. To edit the `php.ini` in IntelliJ IDEA, click Open in Editor. – (Reload): Click this button to check that the specified PHP home directory actually contains a PHP executable file. If no PHP executable is detected at the specified location, IntelliJ IDEA displays the corresponding error message. – (Show phpinfo): Click this button to have IntelliJ IDEA display a separate information window where you can examine the installation details and view the list of loaded extension and configured options. Please note that the options specified in the Configuration Options field of the Interpreters dialog box are not listed. – Debugger extension: in this text box, specify the location of the Xdebug extension to enable IntelliJ IDEA to activate it when necessary. Starting with version 2016.2, IntelliJ IDEA supports the On-Demand mode where you can disable debugger for your global PHP installation, and have it enabled automatically on demand only when you are debugging your command-line scripts or when you need code coverage reports. This lets your command line scripts (including Composer and unit tests) run much faster. – Configuration Options: Use this text box to customize the configuration settings of the installation by composing a string of configuration directives to be passed through the -d command line option and thus add new entries to the `php.ini` file. Click the Browse button to open the Configuration Directives dialog box and create a list of new `php.ini` entries there.

Configuration directives dialog

Item	Tooltip and Shortcut	Description
Name		In this text box, type the name of the new entry.
Value		In this text box, type the value of the new entry.
+	Add `Alt+Insert`	Click this button to have a new line added to the list and specify the name and value of a new entry there.
−	Remove `Alt+Delete`	Click this button to remove the selected entry from the list.
	Up / Down `Alt+Up` / `Alt+Down`	Use these buttons to move the selected entry up or down lin the list. The order of entries in the list determine the order in which they are passed through the -d command line option .

Configure Remote PHP Interpreter

File | Settings | Languages and Frameworks | PHP for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | PHP for OS X

`Ctrl+Alt+S`

The dialog box is available only when the PHP Remote Interpreter plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The dialog box opens when you click the Add toolbar button + in the left-hand pane of the Interpreters dialog box and choose Remote... from the drop-down menu.

Use this dialog box to configure access to PHP engines installed on remote hosts or in development environments set up in Vagrant instances.

Item	Description

SSH Credentials	Choose this option to configure access to a PHP interpreter on a remote host through SSH credentials. In the fields of the dialog box, specify the following:
	– Host: in this field, type the name of the host on which the interpreter is installed.
	– Port: in this field, type the port which the SSH server on the remote host listens to. The default port number is 22.
	– User name: in the field, type the user name under which you are registered on the SSH server.
	– Auth type: from this drop-down list, choose the authentication method.
	  – To access the host through a password, choose Password from the Auth type drop-down list and type the password.
	  – To access the host through a pair of SSH keys, choose Key pair, then specify the path to the file where your private key is stored or the passphrase if you have configured it during the generation of the key pair.
	
	To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . IntelliJ IDEA first attempts to retrieve path mappings itself by processing all the available application-level configurations. If IntelliJ IDEA finds the configurations with the same host as the one specified above, in the Host field, the mappings from these configurations are merged automatically. If no configurations with this host are found, IntelliJ IDEA displays an error message informing you that path mappings are note configured.
	
	To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the server access configuration in question, switch to the Mappings tab, and map local folders to folders on the server as described in Creating a Remote Server Configuration , section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.

Vagrant	This option is available only when the Vagrant repository plugin is installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Choose this option to configure access to a PHP interpreter installed in a Vagrant instance using your Vagrant credentials. Technically, it is the folder where the VagrantFile configuration file for the desired environment is located. Based on this setting, IntelliJ IDEA detects the Vagrant host and shows it as a link in the Vagrant Host URL read-only field. To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . IntelliJ IDEA evaluates path mappings from the `VagrantFile` configuration file.
Deployment Configuration	This option is available only when the Remote Hosts Access plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins . Choose this option to configure access to a PHP interpreter on a remote host using a server access configuration. This option is available only if you have at least one server access configuration of the type SFTP, see Creating a Remote Server Configuration . From the Deployment Configuration drop-down list, choose the server access configuration of the type SFTP according to which you want IntelliJ IDEA to connect to the target host. If the settings specified in the chosen configuration ensure successful connection, IntelliJ IDEA displays the URL address of the target host as a link in the Deployment Host URL field. To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . By default, IntelliJ IDEA retrieves path mappings from the chosen server access (deployment) configuration. If the configuration does not contain path mappings, IntelliJ IDEA displays the corresponding error message. To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the relevant server access configuration, switch to the Mappings tab, and map the local folders to the folders on the server as described in Creating a Remote Server Configuration , section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.
PHP Interpreter Path	In this field, specify the location of the PHP executable file in accordance with the development environment configured on the selected remote host. By default IntelliJ IDEA suggests the `/usr/bin/php` folder. To specify another folder, click the Browse button ... and choose the relevant folder in the dialog box that opens. Note that the PHP home directory must be open for edit. When you click OK, IntelliJ IDEA checks whether the PHP executable is actually stored in the specified folder. – If no PHP executable is found, IntelliJ IDEA displays an error message asking you whether to continue searching or save the interpreter configuration anyway. – If the PHP executable is found, you return to the Interpreters where the installation folder and the detected version of the PHP interpreter are displayed.

WordPress

File | Settings | Languages and Frameworks | WordPress for Windows and Linux

IntelliJ IDEA | Preferences | Languages and Frameworks | WordPress for OS X

`Ctrl+Alt+S`

The page is available only when the WordPress Support plugin is enabled. The plugins are bundled with IntelliJ IDEA and activated by default. If the plugins are not activated, enable them on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Use this page to enable the use of the WordPress Content Management System through the IntelliJ IDEA user interface. To use WordPress in the command line mode, configure it as a command line tool in Command Line Tool Support: WP-CLI .

Item	
Enable WordPress Integration	Select this check box to enable the use of the specified WordPress installation through the IntelliJ IDEA interface.
WordPress Installation Path	In this text box, specify the folder where WordPress is installed. This folder should contain the `wp-admin` and `wp-includes` subdirectories.

Tools

File | Settings | Tools for Windows and Linux

for OS X

Ctrl+Alt+S

When you select the Tools category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- Web Browsers
- File Watchers
- External Tools
- Terminal
- Command Line Tool Support
- Database
- SSH Terminal
- Diagrams
- Diff & Merge
- Docker Machine
- Remote SSH External Tools
- Server Certificates
- Settings Repository
- Startup Tasks
- Tasks
- Vagrant
- XPath Viewer

Web Browsers

File | Settings | Tools | Web Browsers for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Web Browsers for OS X

Ctrl+Alt+S

On this page

- Integrate installations of Web browsers with IntelliJ IDEA, activate or deactivate launching Web browsers from IntelliJ IDEA .
- Specify whether a browser will be launched by running its executable file or through the default system command .
- Appoint the default IntelliJ IDEA browser in which IntelliJ IDEA will open HTML and JSP files upon request by default, that is, when no browser is specified explicitly .

Browsers

In this section, specify which browsers will be available for previewing HTML or JSP output. The section shows a predefined list of browsers, possibly extended with previously configured custom browser installations. Each browser is presented as a separate table row.

IntelliJ IDEA is shipped with a predefined list of most popular browsers which you may like to install and use. The items are added to the list in advanced and are not based on the information on actually installed browsers. IntelliJ IDEA presumes that you install browsers according to a standard procedure. Based on this assumption, each browser in this predefined list is assigned an alias which stands for the path to its executable file, as IntelliJ IDEA supposes it to be. If in your actual browser installation the path to the executable file is different, you need to specify it explicitly in the Path field.
In addition to the predefined browsers, you can configure as many custom browser installations as you need using the controls on the toolbar.

Column	Description
Active	Select this check box to enable the use the respective browser from IntelliJ IDEA. The browser will be added to the context menu of the Open in Browse menu item and its icon will be displayed in the Browsers pop-up toolbar. If this check box is cleared, the corresponding browser icon will not appear in the icons toolbar or pop-up menu.
Name	In this column, specify the browser name.
Family	In this column, specify the family to which the browser belongs.
Path	In this column, specify the path to the executable. If the browser was installed according to a standard installation procedure, most likely the alias shown in the Path field points at the right location of the executable file. To specify the path explicitly, click ... in the Path field and choose the actual location of the executable file in the dialog box that opens. In the dialog that opens , choose the path to the executable file of the corresponding browser.

Toolbar

Item	Description
+	Click this button to add a custom browser to the list.
−	Click this button to delete the selected customer browser from the list. Note that you cannot delete the browsers from the predefined list.
	Click this button to specify a custom profile for Firefox or a browser of the Chrome family. The button is available only when Firefox and Chrome are selected. In the Firefox Settings dialog box, specify the Firefox browser profile to use for previewing output: – Path to "profiles.ini": in this text box, specify the location of the `profiles.ini` file, which determines the Firefox profile to be used. – Profile: from this drop-down list, select the desired predefined profile to use. Learn more at Firefox browser profile . In the Chrome Settings dialog box: – Command line options: In this text box, enter the command line options to launch an instance of Chrome. If you need more space, click , or press Shift+Enter to open the editor box. Learn more about Chrome command line options by opening chrome://flags in Chrome. – Use custom profile directory: Select this check box to define a user-specific Chrome profile to use and specify the location of the `chrome-user-data` directory, which determines the Chrome profile to be used. Learn more about Chrome profiles at Multi-profiles .
↑ ↓	Use these buttons to move the selected browser up or down in the list. The order of browsers is important for rendering external resources and previewing files with Web contents.
	Click this button to create a copy of the selected browser.

Default Browser

In this section, specify the default IntelliJ IDEA browser that will be used by default for rendering external resources and previewing files with Web contents. This browser will be referred to as Default in the context menu when you choose View | Open in Browser on the main menu or Open in Browser on the context menu of a file.

Item	Description
Default browser	Select the default browser from the drop-down list. The possible options are: – System default: Select this option to accept your operating system default Web browser as default for IntelliJ IDEA. – First listed: Select this option to have IntelliJ IDEA launch the first browser in the list. Change the order or browsers using the ↑ and ↓ icons on the toolbar. – Custom path: Select this option to specify another Web browser as default for IntelliJ IDEA. Type the path to the executable file of the browser or click ... and select the path in the dialog box that opens.
Show browser popup in the editor	– If this check box is selected, the popup window with the enabled browsers appears in the HTML or JSP files. – If this check box is not selected, this popup window does not show up, thus helping you read or edit code.

File Watchers

File | Settings | Tools | File Watchers for Windows and Linux

for OS X

Ctrl+Alt+S

The page is available when the File Watchers plugin is enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Use this page to create project File Watchers based on predefined IntelliJ IDEA File Watcher templates and thus enable transpilation in the project.

IntelliJ IDEA supports integration with various third-party transpilers that run in the background and perform the following:

- Translate Less, Sass, and SCSS source code into CSS code.
- Translate TypeScript and CoffeeScript source code into JavaScript code, possibly also creating source maps to enable debugging.
- Compress JavaScript and CSS code.

Note that IntelliJ IDEA does not contain built-in transpilers but only supports integration with the tools that you have to download and install outside IntelliJ IDEA.

In IntelliJ IDEA, these transpiler configurations are called File Watchers. For each supported transpiler, IntelliJ IDEA provides a predefined File Watcher template. Predefined File Watcher templates are available at the IntelliJ IDEA level. To run a transpiler against your project files you need to create a project-specific File Watcher based on the relevant template, at least, specify the path to the transpiler to use on your machine.

You can download a transpiler of your choice and set it up as a File Watcher. However, in this case no predefined template is available so you will have to specify all the settings manually.

To be applicable, a File Watcher must be enabled by selecting the check box next to it on the File Watchers page of the Settings dialog box, see Enabling and Disabling File Watchers . After that the File Watcher will be invoked automatically upon any changes made to the source code or upon save, depending on whether the Immediate File Synchronization check box is selected or cleared, see New Watcher Dialog .

The output of a File Watcher is stored in a separate file. The predefined templates suggest the type of the file depending on the transpiler type. By default the output file is created in the same folder as the input file when the File Watcher is invoked for the first time, whereupon this file is only updated. You can customize all these settings during File Watcher creation.

JavaScript files generated by File Watchers are excluded from code completion and refactoring.

In the Project tree view, the output file is shown under the original file which is shown as a node. This is done to improve visibility so you can easier locate necessary files.

File watchers have two dedicated code inspections:

- The File watcher available inspection is invoked in every file that is recognized as subject for applying a predefined file watcher (Sass, Less, SCSS, TypeScript, or CoffeeScript). If none of the applicable predefined File Watchers is associated with the current project, IntelliJ IDEA suggests to add one.
- The File watcher problems inspection is invoked by a running File Watcher and highlights errors specific for it.

The File Watchers page consists of two parts:

- A list of File Watchers available in the current project. To activate a File Watcher, select the check box next to it. If an error occurs while a File Watcher is running, the File Watcher is automatically disabled.
- A toolbar to manage this list.

ItemTooltip/ and shortcut	Description
Add Alt+Insert	Click this button to open the Choose template pop-up list and choose the relevant type of File Watcher. After that IntelliJ IDEA opens the New Watcher dialog box for customizing the predefined File Watcher according to the settings of the current project.
Edit Enter	Click this button to update the settings of the selected File Watcher in the Edit Watcher dialog box. The update is applied to the current project File Watcher only, it does not affect the predefined IntelliJ IDEA-level template.
Remove Alt+Delete	Click this button to remove the selected File Watcher. The File Watcher is no longer applied to the files in the current project. Note that this action does not affect the corresponding predefined template which is still available at the IntelliJ IDEA level.
Up(Ctrl+Alt+Up) Down (Ctrl+Alt+Down)	Use these buttons to change the order of File Watcher in the list. This determines the order of launching File Watchers, if more than one are enabled.
Copy	Use this button to create a copy of the selected file watcher.
Import	Click this button to import an existing file watcher and add it to the list of available file watchers.
Export	Click this button to export the selected watchers to `watchers.xml` file, located under the user's home.

New Watcher Dialog

File | Settings | Tools | File Watchers for Windows and Linux

IntelliJ IDEA | Preferences | Tools | File Watchers for OS X

Ctrl+Alt+S

The dialog box opens when you click the Add + or Edit button on the File Watchers page . Use the dialog box to create a project File Watcher based on a predefined IntelliJ IDEA File Watcher template or to edit an existing project File Watcher.

Each template contains the settings that are optimal for the selected transpiling tool. So in most cases, all you need is specify the path to the transpiler executable.

On this page:

– Name and Description Area
– Options Area
– Watcher Settings Area
– Examples of customizing the behaviour of a transpiler

Name and Description Area

Item	Description
Name	In this text box, type the name of the File Watcher. By default, IntelliJ IDEA suggests the name of the selected predefined template.
Description	In this text box, describe the functionality of the File Watcher. Predefined templates already contain general information regarding the accepted input and produced output. Expand or update the description, if, for example, you have two File Watcher that apply to the same source code but differ from each other in some configuration settings.

Options Area

In this area, configure the behaviour of the File Watcher.

Item	Description
Output Filters	Click this button to open the Output Filters dialog where you can manage the list of filters to distinguish the output of the File Watcher from other output. These filters make the basis for: 1. Displaying paths to the File Watcher output files as links in error and other messages and logs. When you click such link, the corresponding file is opened in the editor. 2. Error highlighting in the output files.
Immediate file synchronization	– Select this check box to have the File Watcher invoked as soon as any changes are made to the source code. – If the check box is cleared, the File Watcher will start upon save (File | Save All, ?) or when you move focus from IntelliJ IDEA (upon frame deactivation).
Track only root files	In the TypeScript file watcher, the check box has been renamed to Compile main files only. When the File Watcher is invoked on a file, IntelliJ IDEA detects all the files in which this file is included. For each of these files, in its turn, IntelliJ IDEA again detects the files into which it is included. This operation is repeated recursively until IntelliJ IDEA reaches the files that are not included anywhere within the specified scope . These files are referred to as root files (do not confuse with content roots). – When this check box is selected, the File Watcher runs only against the root files. – When the check box is cleared, the File Watcher runs against the file from which it is invoked and against all the files in which this file is included recursively within the specified scope. This option is available only for Babel, Closure Compiler, Compass, Jade, Less, Sass/ SCSS, Stylus, Typescript, UglifyJS, and YUI Compressor JS.
Show console	Use this drop-down list to define when you want the File Watcher open its dedicated console with messages. The available options are: – Always: when this option is chosen, the File Watcher opens the console when it starts. – Error: when this option is chosen, the File Watcher opens the console only if any errors occur in transpilation. – Never: choose this option to suppress opening the console under any circumstances.
Check file for syntax errors	– Select this check box to have the File Watcher ignore update, save, and change focus events in files that are syntactically invalid. – If this check box is cleared, the File Watcher starts anyway upon update, save, or frame deactivation, depending on the status of the Immediate file synchronization check box.

Watcher Settings Area

In this area, configure interaction with the transpiling tool: specify the executable file to use, the arguments to pass to it, and customize the default template settings for input and output.

Item	Description
File type	Use this drop-down list to specify the expected type of input files. The File Watcher will consider only files of this type as subject for analyzing and processing. File types are recognised based on associations between file types and file extensions . By default, the field shows the file type in accordance with the chosen predefined File Watcher template.
Scope	Use this drop-down list to define the range of files the File Watcher can be applied to. Changes in these files will invoke the File Watcher either immediately or upon save or frame deactivation, depending on the status of the Immediate file synchronization check box. You can choose one of the predefined scopes from the drop-down list: – Project Files: all the files within the project content roots (see Content Root and Configuring Content Roots). – Project Production Files: all the files within the project content roots excluding test sources. – Project Test Files: all the files within the project test source roots. – Open Files: all the files that are currently opened in the editor.

VCS Scopes: these scopes are only available if your project is under version control .

- Changed Files: all changed files, that is, all files associated with all existing changelists .
- Default: all the files associated with the changelist `Default` .

Alternatively, click the Browse button and configure a custom scope in the Scopes dialog box that opens.

For more details on scopes, see the pages Scopes and Scopes dialog .

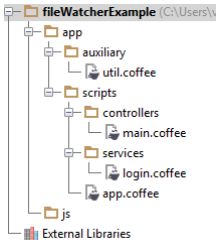
The Scope setting overrides the Track only root files check box setting: if a dependency is outside the specified scope, the File Watcher is not applied to it.

Program	In this text box, specify the location of the transpiler's executable file (`.exe` , `.cmd` , `.bat` , or other depending on the specific tool. `.jar` archives are also acceptable but defining `PATH` variables for them is not supported.) Do one of the following: – Type the path explicitly. – Click the Browse button … to open the Select Path dialog box and navigate to the desired location. – Click the Insert Macro button to open the Macros dialog box where you can select the relevant macro from the list.
Arguments	In this text box, define the arguments to pass to the transpiler and thus influence its behaviour. Among other cases, use this text box to change the default output location, that is, specify a custom location where you want the transpiler to store the files generated during transpilation. Do one of the following: – Type the list of arguments in the text box. – Click the Insert Macro button to open the Macros dialog box where you can select the desired macro from the list. When specifying the arguments, follow these rules: – Use spaces to separate individual arguments. – If an argument includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an argument includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value"\` .
Working directory	In this text box, specify the directory to which the transpiler will be applied. Because the tool is always invoked in the context of a file, the default working directory is the directory of the current file. This setting is specified in all predefined templates through a macros `$FileDir$` . To update this default settings, do one of the following: – Type the path explicitly in the text box. – Click the Browse button … to open the Select Path dialog box and navigate to the desired location. – Click the Insert Macro button to open the Macros dialog box, where you can select the desired macro from the list. Tip If the field is left blank, IntelliJ IDEA uses the directory of the file where the File Watcher is invoked.
Environment variables	Use this text box to specify a the `PATH` variable for a tool that is required for starting the transpiler but is not referenced in the path to it.
Output paths to refresh	In this text box, specify the files where the transpiler stores its output: the resulting source code, source maps, and dependencies. In other words, tell IntelliJ IDEA where it should search for the files generated through transpilation. Please note, that changing the value in this text box does not make the transpiler store its output in another location. To do that, specify the desired output location in the Arguments text box. Do one of the following: – Type the output paths manually. Use colons as separators. – Click the Insert Macro button to open the Macros dialog box, where you can select the desired pattern from the list.
Create output from stdout	– When this check box is selected, IntelliJ IDEA reads the native transpiler's output (`standard output stream (stdout)`) and generates the resulting files from it. – When the check box is cleared, the transpiler writes its output directly to the files specified in the Output paths to refresh field. Tip Some transpilers generate a `standard output stream (stdout)` file, others do not, which may lead to errors. Therefore it is strongly recommended to preserve the default setting.

Examples of customizing the behaviour of a transpiler

Any transpiler is an external, third-party tool. Therefore the only way to influence a transpiler is pass arguments to it just as if you were working in the command line mode. These arguments are specific for each tool. Below are two examples of customizing the default output location for the CoffeeScript transpiler.

Suppose, you have a project with the following folder structure:



By default, the generated files will be stored in the folder where the original file is. You can change this default location and have the generated files stored in the `js` folder. Moreover, you can have them stored in a flat list or arranged in the folder structure that repeats the original structure under the `app` node.

– To have all the generated files stored in the output `js` folder without retaining the original folder structure under the `app` folder:

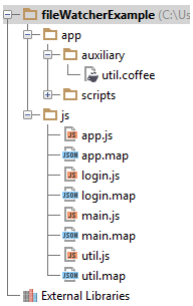
1. In the Arguments text box, type:

```
--output $ProjectFileDir$\js\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir$\js\$FileNameWithoutExtension$.js:$ProjectFileDir$\js\$FileNameWithoutExtension$.map
```

As a result, the project tree looks as follows:



- To have the original folder structure under the `app` node retained in the output `js` folder:

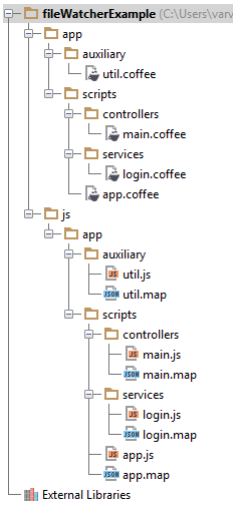
1. In the Arguments text box, type:

```
--output $ProjectFileDir$\js\$FileDirRelativeToProjectRoot$\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir$\js\$FileDirRelativeToProjectRoot$\$FileNameWithoutExtension$.js:$ProjectFileDir$\js\$FileDirRelativeToI
```

As a result, the project tree looks as follows:



External Tools

File | Settings | Tools | External Tools for Windows and Linux

IntelliJ IDEA | Preferences | Tools | External Tools for OS X

Ctrl+Alt+S

Define third-party standalone applications (code generators and analyzers, pre- and post- processors, database utilities, etc.) as external tools to be able to run them from IntelliJ IDEA.

You can pass contextual information (like the currently selected file, or your project source path) to the external tools, view the tool output, and more.

The tools defined on this page appear as commands in the Tools menu and in various context menus. They can also be assigned keyboard shortcuts (see Configuring Keyboard Shortcuts).

Toolbar icons

Icon	Description
	Add a definition of an external tool. (The Create Tool dialog will open.)
	If an individual tool is selected: delete the tool definition. If a tool group is selected: delete the definitions of all the tools within the selected group.
	Edit the definition of the selected tool. (The Edit Tool dialog will open.)
	Move the selected tool one line up or down within the group. (The order of tools defines the order of items in corresponding menus.)
	Create a copy of the selected definition and then edit that copy. (The Copy Tool dialog will open.)

Check boxes

Use the check boxes to enable or disable the tools and the tool groups. The items that are not currently selected are not available in the Tools and context menus.

Create/Edit/Copy Tool Dialog

From the External Tools page :   or

Edit the settings for your external tool.

Item	Description
Name	The name of the tool that appears as a command name in the Tools menu and the context menus. See also, Show in .
Group	The group the tool belongs to. The tool groups correspond to submenus in the Tools menu and the context menus. Select an existing group from the list or type the name for a new group.
Description	The tool description (optional).
Options	
Synchronize files after execution	Make IntelliJ IDEA aware of changes in the file system when the tool completes its execution.
Open console	Open the console for viewing the tool output such error messages, etc.
Output Filters	Open the Output Filters dialog to manage the output filters associated with the tool. (The output filters are used to turn absolute file paths and line numbers in the tool output into hyperlinks. You'll be able to use those links to open the corresponding files in the editor.)
Show console when a message is printed to standard output stream	Make the output console active and bring it forward when the corresponding event occurs.
Show console when a message is printed to standard error stream	The same as the previous option but for stderr.
Show in	Specify in which menus the command for running the tool should be included. Main menu means the Tools menu. The rest of the options correspond to context menus in various places.
Tool settings	
Program	The path to the executable file to be run. Use to select the file, Insert macro to open the Macros dialog to select a macro. (Macros are resolved at runtime and let you specify context information such as currently selected file, your project source paths, etc.)
Parameters	The parameters to be passed to the program the way you'd specify them on the command line. Use Insert macro to open the Macros dialog to select a macro. When specifying the parameters, follow these rules: – Use spaces to separate individual parameters. – If a parameter includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Working directory	The path to the current working directory for the program. Use to select the directory, Insert macro to open the Macros dialog to select a macro.

Add/Edit Filter Dialog

From the Output Filters dialog : + or

Specify the filter for transferring absolute file paths and line numbers in the tool output into hyperlinks.

Item	Description
Name	The name of the filter.
Description	The filter description (optional).
Regular expression to match output	The text pattern to be matched against the tool output to identify linkable references. The pattern can include text and the following placeholder variables: – `$FILE_PATH$` - the portion of the output that corresponds to an absolute path to a source file. Required. – `$LINE$` - a line number reference. – `$COLUMN$` - a column reference. The variables are inserted by right-clicking the field and then selecting the necessary item from the list that is shown. Example If your tool outputs the lines similar to `Error parsing C:\Demos\src\converter\MetersToInches.xml:103 Missing Closing Tag` the pattern `$FILE_PATH$:$LINE$` will turn `C:\Demos\src\converter\MetersToInches.xml:103` into a hyperlink to the line number `103` in the file `MetersToInches.xml`.

Output Filters Dialog

From the Create/Edit/Copy Tool dialog : Output Filters

This dialog lets you manage the output filters associated with an external tool. (The output filters are used to turn absolute file paths and line numbers in the tool output into hyperlinks.)

Item	Description
+	Create a new filter. (The Add Filter dialog will open.)
−	Delete the selected filter.
	Edit the selected filter. (The Edit Filter dialog will open.)
	Move the selected filter one line up or down in the list. (For each line in the output, the first matching filter is used.)

Macros Dialog

From the Create/Edit/Copy Tool dialog : Insert macro

Select the macro to be inserted.

Item	Description
Macros	The list of available macros with their descriptions.
Macro preview	The value of the selected macro in the current context.

Terminal

File | Settings | Tools | Terminal for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Terminal for OS X

Ctrl+Alt+S

Use this dialog to specify settings for the embedded local terminal.

Prerequisites

Before you start working with terminal, make sure that the Terminal plugin is enabled. The plugin is bundled with IntelliJ IDEA and is activated by default. If it is not, enable the plugin as described in Enabling and Disabling Plugins .

Item	Description
Shell path	Specify the shell that will run by default.
Tab name	Specify the default name of a new session tab. Note that a session tab can be renamed.
Close session when it ends	If this check box is selected, the current session ends automatically, when the corresponding process ends (for example, by kill).
Audible bell	If this check box is selected, the console plays the bell sound on incoming escape sequence.
Mouse reporting	If this check box is selected, the embedded local terminal supports mouse pointer.
Copy to clipboard on selection	If this check box is selected, the text selected in the Terminal, is automatically passed to the system clipboard.
Paste on middle mouse button click	If this check box is selected, you can paste the clipboard contents by clicking the middle mouse button.
Override IDE shortcuts	If this check box is selected (this is the default setting), the Terminal tool window handles keyboard shortcuts differently from IntelliJ IDEA. If this check box is cleared, the IntelliJ IDEA key bindings are used.

Command Line Tool Support

File | Settings | Command Line Tool Support for Windows and Linux

IntelliJ IDEA | Preferences | Command Line Tool Support for OS X

Ctrl+Alt+S

This page appears in the Settings/Preferences dialog box, when the Command Line Tool Support plugin is downloaded, installed and enabled. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . The page shows a list of all PHP-specific and custom command line tools that are currently available at the IntelliJ IDEA level.

Tools with inconsistencies in the definition files are marked with the Invalid description icon .

Use this page to enable integration with third-party PHP command line tools and define your own ones.

Item	Tooltip and shortcut	Description
Table of available tools		
Enabled		When the check box in this column is selected, the commands defined within the corresponding command line tool can be executed from IntelliJ IDEA.
Alias		In a text box in this column, specify the character string to use in command calls instead of the full path to the tool. For example, by default, IntelliJ IDEA assigns the following aliases: – `s` for Symfony. – `zf` for Zend Framework. – `c` for Composer.
Tool Path		In a text box in this column, specify the location of the tool's executable file.
Type		A read-only field in this column shows the official name of the third party command line tool. 1. The column is available only if the Show tool type check box is selected. 2. For custom command line tools, the fields in this column are empty.
Toolbar		
	Add Alt+Insert	Click this button to open the Command Line Tools dialog box and select the type of command-line tool to enable integration with. The available options are: – Custom Tool – Zend Framework 1 Tool – Zend Framework 2 Tool – Symfony – Composer – Drush – Tool based on Symfony Console
	Remove Alt+Delete	Click this button to have the selected tool removed from the list.
	Edit	Click this button to change the definition of the selected tool.
	Reload command list from executable	Click this button to have IntelliJ IDEA refresh the list of commands of the selected tool.
	Open definition in editor	Click this button to have IntelliJ IDEA open the `.xml` file with commands of the selected tool in the editor.
Show tool type		If this check box is selected, the Type column is added to the table of available tools.
Show console in		In this area, specify where you want to enter commands. The available options are: – Pop-up - choose this option to have the Command Line Tools Input pane opened in a separate pop-up window and type commands there. – Tool window - choose this option to enter commands in the Input field in the bottom of the dedicated Command Line Tools Console tool window.
Console encoding		From this drop-down list, choose the character set to show the tool's output in the Command Line Tools Console Tool Window .

Database

File | Settings | Tools | Database for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Database for OS X

From the database , Hibernate and JPA consoles:

From the table editor : | Settings

The Database and subordinate pages contain the settings related to working with databases and SQL.

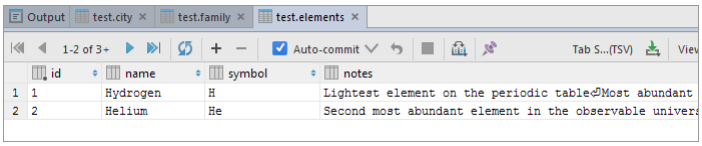
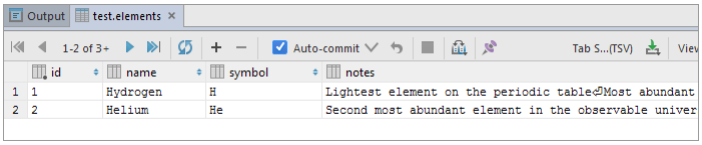
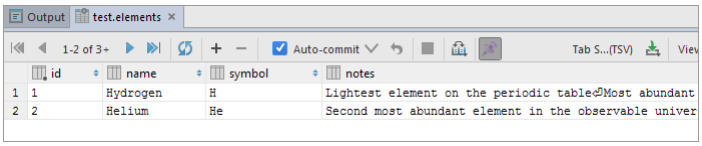
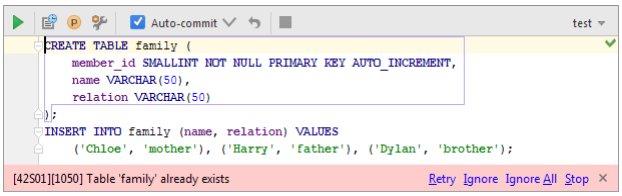
- Console
- Execute in Console
- Quick Documentation

- Data Views
- User Parameters
- CSV Formats

See also, SQL Dialects .

Console

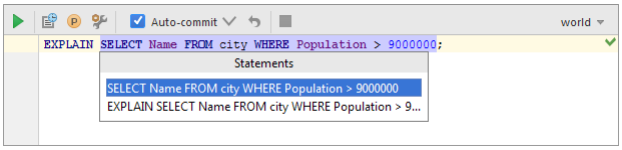
The settings in this section relate to showing various information in database consoles .

Item	Description
Show query results in new tab	You can select to view query results on individual tabs, or on one and the same tab. If the check box is selected, a new tab with the query result will open each time you run a query (`SELECT`). In this way, you can keep the results of all the queries that you have run.  If this check box is not selected, the same tab is used to show your query results. When you run a query, the information on the tab is updated to show the result, and a new tab doesn't open.  In this case, when you get the result that you want to keep, you can pin the tab to the tool window (on the toolbar or Pin Tab in the context menu for the tab). 
Show error notifications in editor	If this check box is not selected, the information about the errors is shown only in the output pane. If, in addition, you want the error notification bar to appear in the input pane, select the check box. The error notification bar may be particularly useful when running sequences of SQL statements. If an error occurs in such cases, the error notification bar lets you select how to react.  For more information, see Using the error notification bar.
Always review parameters before execution	When you run a statement with parameters, IntelliJ IDEA memorizes the parameter values. Each next time you execute the statement: – If this option is on, IntelliJ IDEA shows you the last used parameter values so that you can change them before actually running the statement. – If this option is off, IntelliJ IDEA executes the statement right away without showing you the parameter values.

Execute in Console

This section contains the options for the Execute command.

Item	Description
When inside a statement execute	If the cursor is inside a statement, the following options are available: – Ask what to execute. A list of statements that you can run is shown and you can select the statement or statements.



- Smallest statement. The smallest of the possible statements is executed. For example, when the cursor is inside a subquery, the subquery is executed.
- Largest statement. The largest possible statement is executed. For example, when the cursor is inside a subquery, an outer statement is executed.
- Largest statement or batch. For Transact-SQL (SQL Server and Sybase), the current batch of statements is executed. For all other dialects - the same as the previous option.
- Whole script. All the statements are executed.

otherwise execute	If the cursor is outside of a statement, e.g. on a blank line or within a comment, the following options are available: – Nothing. None of the statements is executed. – Whole script. All the statements are executed. – Everything below caret. All the statements after the cursor position are executed.
for selection execute	If something is currently selected, the following options are available: – Exactly as one statement. Exactly what is selected is executed as a single statement. – Exactly as statements. Exactly what is selected is executed. If the selection contains more than one statement, the statements are executed as separate statements. – Smart expand to script. If there is at least one statement border within the selection, the selection is expanded to form a sequence of valid statements. This sequence is then executed. Otherwise, precisely what is selected is executed.

Quick Documentation

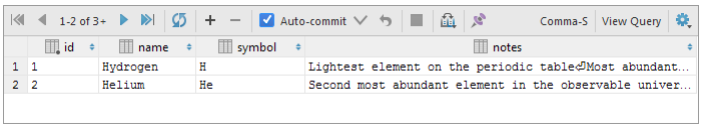
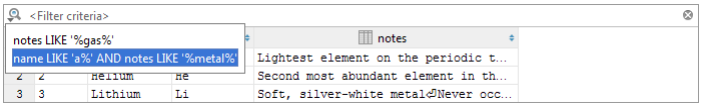
Item	Description
Allow database access	Let IntelliJ IDEA access the database when generating quick documentation. For example, for a table: – If this option is on, first rows are shown, and the table definition (`CREATE TABLE`) is retrieved from the database. – Otherwise, the data are not shown, and an auto-generated table definition is presented. Generally, when the option is on, you get more info but it takes more time to get the info.
Number of preview rows	The number of rows to be shown for a table in the quick documentation view.

Data Views

File | Settings | Database | Data Views for Windows and Linux

IntelliJ IDEA | Preferences | Database | Data Views for OS X

The settings on this page define how table data are shown in the database , Hibernate and JPA consoles, table editor and quick documentation preview.

Item	Description
Result set page size	The number of table rows to be shown at a time, on one "page". Here is an example when this number is set to 2:  If you don't want to limit the number of rows displayed simultaneously, specify zero (0).
Result set prefetch size	Data from databases are retrieved in chunks. The number in this field defines the number of rows in such chunks. A bigger number means fewer IDE - DB round trips but more memory for storing a chunk.
Filter history size	The number of most recently used filtering conditions to memorize for a table in the table editor. Here is an example when this number is set to 2. (The filter history box contains two most recently used conditions.) 
LOB length limit	The maximum size of a binary large object to be loaded in bytes.

User Parameters

File | Settings | Tools | Database | User Parameters for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Database | User Parameters for OS X

Generally, only the question mark (`?`) is treated as a parameter in SQL statements. On this page, you can specify which other characters and their sequences should be treated as parameters, and in which places.

The pattens for SQL parameters are specified by means of regular expressions.

Item	Description
Enable in console and SQL files	If the check box is selected, the parameter patterns are applied to SQL (in SQL files and database consoles). The usage scope, if necessary, may be limited at the level of individual patterns . If this check box is not selected, the patterns are not used in SOL files and consoles irrespective of which usage scope is specified for individual patterns.
Enable in string literals with SQL injection	If the check box is selected, the parameter patterns are applied to string literals injected with SQL in the languages other than SQL. (See Using Language Injections .) The usage scope, if necessary, may be limited at the level of individual patterns . If this check box is not selected, the patterns are not used in string literals irrespective of which usage scope is specified for individual patterns.
Parameter patterns	The table shows the parameter patterns and their usage scopes. The patterns are specified using regular expressions. Values in parentheses are treated as parameter names. The patterns available initially have the following meanings: – `\?(\d+)` - a question mark followed by one or more digits, e.g. `?69` in which case `69` would be the parameter name. – `:(\w+)` - a colon followed by one or more word characters, e.g. `:x` , `:value` , `:parameter_1` . – `%w+` - `%` followed by one or more word characters, e.g. `%xyz` . – `\$\{([^$\{\}]*)\}` - `$` , then `{` , then any character except `$` , `{` or `}` zero or more times, then `}` , e.g. `${}` , `${value}` . – `\$\(([^\)]+)\)` - `$` , then `(` , then any character except `)` one or more times, then `)` , e.g. $(x). – `\$(\w+)\$` - `$` , then one or more word characters, then `$` again, e.g. `$x1$` . – `\#(\w+)\#` - `#` , then one or more word characters, then `#` again, e.g. `#field_3#` . Use + (`Alt+Insert`), – (`Alt+Delete`), ↑ (`Alt+Up`) and ↓ (`Alt+Down`) to add, delete and reorder the patterns. To edit a pattern or its usage scope, click the pattern and use the following controls: – In scripts. Clear this check box if the pattern shouldn't be used in SQL files and database consoles. – In literals. Clear this check box if the pattern shouldn't be used in string literals injected with SQL. – All (the link text may be different). Click the link and deselect the languages in which the pattern shouldn't be used.

CSV Formats

File | Settings | Tools | Database | CSV Formats for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Database | CSV Formats for OS X

This page contains the settings for converting table data into delimiter-separated values formats (e.g. CSV, TSV) and vice versa.

When working on the conversion settings, use the preview in the right-hand part of the page.

Item	Description
Formats	The list of the available delimiter-separated values formats is shown. Each format is a named set of corresponding conversion settings. Select the format whose settings you want to view or edit. Use +, −, ↑ and ↓ to create, delete and reorder the formats; to create a copy of the selected format.
Value separator	Select or type the character for separating individual values.
Row separator	Select or type the character for separating rows.
Null value text	The text to be used as a value if a cell contains `null` (an unknown value).
Add row prefix/suffix	Row prefix and suffix are character sequences which in addition to the row separator indicate the beginning and end of a row. If necessary, click the link and specify the row prefix and suffix in the fields that appear.
Quotation	Each line in the area under Quotation is a quotation pattern (see Quote values). A quotation pattern includes: – The left quotation character, the one inserted before a value. – The right quotation character, the one inserted after a value; usually, the same as the left quotation character. – An escape method or character for the cases when the quotation character is part of a value. E.g. Escape: duplicate means that if a quotation character occurs within a value, it is doubled. (You can specify your own escape character instead.) If there is more than one pattern, the first of the patterns is used. Use +, −, ↑ and ↓ to create, delete and reorder the patterns. To start editing an existing pattern, just click the pattern of interest.
Quote values	Specify in which cases the values should be quoted (i.e. enclosed within quotation characters). – When needed. A value is quoted only if it contains the value and/or the row separator. – Always. Any value is quoted in its text representation.
Trim whitespaces	If this check box is not selected, the Unicode whitespace characters that precede and follow the value separators are treated as parts of the corresponding values. If this check box is selected, the corresponding whitespace characters are ignored or removed.
First row is header	If this check box is selected, the first row is treated as containing column names. The settings that appear under Header Format have the same meanings as the ones above but are applied to the first row.
First column is header	If this check box is selected, the first column is treated as containing row names.

SSH Terminal

File | Settings | Tools | SSH Terminal for Windows and Linux

IntelliJ IDEA | Preferences | Tools | SSH Terminal for OS X

Ctrl+Alt+S

Use this dialog box to appoint the a remote Web server or a Vagrant instance (virtual machine) to access through the SSH terminal, configure connection with the destination environment, and choose the encoding to use in the SSH terminal.

Make sure the SSH Remote Run plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

Item	Description
Connection settings	In this section, appoint the a remote Web server or a Vagrant instance (virtual machine) to access through the SSH terminal and specify where the connection settings should be taken from: 1. Current Vagrant: select this option to have the commands in the SSH Terminal executed on the currently running Vagrant virtual machine. For details, see Vagrant . 2. Deployment server: select this option to have the commands in the SSH Terminal executed on the local or remote Web server accessible through one of the server access configurations . From the drop-down list, choose the server access configuration that specifies the destination environment and the settings to establish connection to it. – Select server on every run: if this option is selected, you will have to choose the desired server access configuration from the pop-up window, every time you choose Tools | Start SSH Session on the main menu. – If the desired server access configuration does not appear in the drop-down list, click the link Configure Servers, and define one in the Deployment page. For details, see Configuring Synchronization with a Web Server .
Default encoding	From this drop-down list, select the desired encoding to be used in the SSH terminal.

Diagrams

File | Settings | Tools | Diagrams for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Diagrams for OS X

`Ctrl+Alt+S`

Use this page to configure the default visibility settings and layout for diagrams.

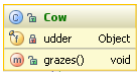
On this page:

– Content pane

– Controls

Content pane

Select the check boxes next to the elements to be shown on diagrams.

Item	Description
Show Difference	
Details	If this check box is selected, all the specified details of the elements will be shown in the UML class diagram for a revision. If this check box is not selected, only node elements will be included in diagram.
Java Class Diagrams	
Class Elements	Select the check boxes to show members (fields, constructors, methods, properties and inner classes) within the node elements:  In diagram, use toolbar buttons , , , , and .
Dependencies	Select the check boxes below to show dependency links in diagram.
DB Diagrams	
Key columns	For the primary key columns to be shown when a diagram opens, select this check box. When viewing a diagram in the editor, use on the toolbar to show or hide the corresponding columns.
Columns	For the columns other than the primary key columns to be shown when a diagram opens, select this check box. When viewing a diagram in the editor, use on the toolbar to show or hide the corresponding columns.
Project Modules	
Libraries	If this check box is selected, libraries will be shown in UML diagrams for modules.
ActionScript/Flex	
Class Elements	Select the check boxes below to show members (fields, constructors, methods, and properties) within the node elements. In diagram, use toolbar buttons , , , and .
JPA ER Diagram	
Properties	If this check box is selected, properties of entity classes are shown when a JPA ER diagram opens. When viewing a diagram in the editor, you can show or hide these properties by using on the toolbar.
Embeddables	If this check box is selected, embeddable objects are shown when a JPA ER diagram opens. When viewing a diagram in the editor, you can show or hide the embeddables by using on the toolbar.
Superclasses	If this check box is selected, superclasses of entity classes are shown when a JPA ER diagram opens. When viewing a diagram in the editor, you can show or hide the superclasses by using on the toolbar.
EJB ER Diagram	
Properties	If this check box is selected, properties of entity beans are shown when an EJB ER diagram opens. When viewing a diagram in the editor, you can show or hide these properties by using on the toolbar.
Embeddables	If this check box is selected, embeddable objects are shown when an EJB ER diagram opens. When viewing a diagram in the editor, you can show or hide the embeddables by using on the toolbar.
Superclasses	If this check box is selected, superclasses of entity beans are shown when an EJB ER diagram opens. When viewing a diagram in the editor, you can show or hide the superclasses by using on the toolbar.
BPMN 2.0 Diagram	
Details	For element details to be shown when a BPMN diagram opens, select this check box.
CDI Dependencies Diagram	
@Inject	If this check box is selected, injection points are shown when a CDI dependency diagram opens. When viewing a diagram in the editor, you can show or hide the injection points by using on the toolbar.
@Produces	If this check box is selected, producer methods and fields are shown when a CDI dependency diagram opens. When viewing a diagram in the editor, you can show or hide the producer methods and fields by using on the toolbar.

@Decorator	If this check box is selected, decorator bean classes are shown when a CDI dependency diagram opens. When viewing a diagram in the editor, you can show or hide the decorators by using on the toolbar.
Spring	
Local context	If this check box is selected, local context will be shown in diagrams. To enable showing local context in the Diagram tab in the editor, click .
Properties	If this check box is selected, property files will be shown in diagrams. To enable showing property files in the Diagram tab in the editor, click .
Spring Integration	
Show Labels	If this check box is selected, labels will be shown in diagrams. To enable showing labels in the Diagram tab in the editor, click .
Spring Web Flow	
Details	If this check box is selected, deatils will be shown in diagrams.
Events	If this check box is selected, events will be shown in diagrams.
Sub Flows	If this check box is selected, Sub Flows will be shown in diagrams.

Note More nodes appear in this pane depending on the installed and enabled plugins.

Controls

Item	Description
Default layout	Select the desired layout from the drop-down list. Node elements in newly created diagrams will be arranged according to the selected layout.
Default scope	Select scope from the drop-down list. Specifying a scope helps you avoid showing in diagram the unnecessary hierarchies. You can define scopes for your project in the Scopes page of the Settings dialog.
Fit content after layout	If this check box is selected, then after applying a layout selected on the diagram context menu, all diagram elements will be resized to fit into the current diagram area. In diagram, use the toolbar button.
Do relayout when new elements were added	If this check box is selected, diagram layout will be performed automatically after adding new elements.
Enable colors	If this check box is selected, relationship links will be shown colored.

Diff & Merge

File | Settings | Tools | Diff & Merge for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Diff & Merge for OS X

Ctrl+Alt+S

On this page, specify the default behavior of the Differences viewer .

Item	Description
Diff	
Context lines	Use the slider to specify the amount of context lines that will not collapse, when in a Differences viewer you click the button to collapse the unchanged fragments .
Go to the next file after reaching last change	If this check box is selected, IntelliJ IDEA will suggest to click F7 / Shift+F7 once more and compare other files
Merge	
Automatically apply non-conflicting changes	If this check box is selected, the interactive merge tool automatically merges all non-conflicting changes. This is an equivalent to clicking in the Merge dialog.

External Diff Tools

File | Settings | Tools | Diff | External Diff Tools for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Diff | External Diff Tools for OS X

Ctrl+Alt+S

If necessary, specify external tools for comparing files and folders, and associated settings.

Item	Description
Use external diff tool	Select this check box to have IntelliJ IDEA use an external tool for comparing files or folders.
Path to executable	In this text box, specify the path to the executable file of the desired external diff tool. Use the Browse button ..., if necessary. This field is only available, when Use external diff tool check box is selected.
Use by default	Select this check box to have IntelliJ IDEA use the specified external tool for comparing files or folders by default. If this check box is not selected, IntelliJ IDEA will use the built-in diff tool . To invoke an external tool, click the button on the toolbar of the Differences viewer . This field is only available, when Use external diff tool check box is selected.
Parameters	Use this field to set the diff tool parameters. Note Note that different diff tools have different parameters. You need to specify all the necessary parameters in proper order. This field is only available, when Use external diff tool check box is selected.
Use external merge tool	Select this check box to have IntelliJ IDEA use an external merge tool. In the text box below, specify the path to the executable file of the desired external tool. Use the Browse button ..., if necessary.
Path to executable	In this text box, specify the path to the executable file of the desired external merge tool. Use the Browse button ..., if necessary. This field is only available, when Use external merge tool check box is selected.
Parameters	Use this field to set the merge tool parameters. Note Note that different merge tools have different parameters. You need to specify all the necessary parameters in proper order. This field is only available, when Use external merge tool check box is selected.

Docker Machine

File | Settings | Tools | Docker Machine for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Docker Machine for OS X

For this page to be available, the Docker integration plugin must be installed.

Specify the settings for working with Docker Machine .

Item	Description
Docker Machine executable	`docker-machine` or an actual path to `docker-machine.exe` (normally located in the Docker Toolbox installation folder). The default setting `docker-machine` is fine if: – The actual name of the executable file is `docker-machine` . – The path to the directory where the file is located is included in the environment variable `Path` .
Detect	Set Docker Machine executable to `docker-machine` , try to detect the executable file and, if a success, show its version.

Remote SSH External Tools

File | Settings | Tools | Remote SSH External Tools for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Remote SSH External Tools for OS X

Define remote applications that require SSH access as external tools to be able to run them from IntelliJ IDEA.

You can pass contextual information (like the currently selected file, or your project source path) to the external tools, view the tool output, and more.

The tools defined on this page appear as commands in the Tools menu and in various context menus. They can also be assigned keyboard shortcuts (see Configuring Keyboard Shortcuts).

Toolbar icons

Icon	Description
	Add a definition of an external tool. (The Create Tool dialog will open.)
	If an individual tool is selected: delete the tool definition. If a tool group is selected: delete the definitions of all the tools within the selected group.
	Edit the definition of the selected tool. (The Edit Tool dialog will open.)
	Move the selected tool one line up or down within the group. (The order of tools defines the order of items in corresponding menus.)
	Create a copy of the selected definition and then edit that copy. (The Copy Tool dialog will open.)

Check boxes

Use the check boxes to enable or disable the tools and the tool groups. The items that are not currently selected are not available in the Tools and context menus.

Create/Edit/Copy Tool Dialog (Remote SSH External Tools)

From the Remote SSH External Tools page : , or

Edit the settings for your external tool.

Item	Description
Name	The name of the tool that appears as a command name in the Tools menu and the context menus. See also, Show in .
Group	The group the tool belongs to. The tool groups correspond to submenus in the Tools menu and the context menus. Select an existing group from the list or type the name for a new group.
Description	The tool description (optional).
Options	
Synchronize files after execution	Make IntelliJ IDEA aware of changes in the file system when the tool completes its execution.
Open console	Open the console for viewing the tool output such error messages, etc.
Output Filters	Open the Output Filters dialog to manage the output filters associated with the tool. (The output filters are used to turn absolute file paths and line numbers in the tool output into hyperlinks. You'll be able to use those links to open the corresponding files in the editor.)
Show console when a message is printed to standard output stream	Make the output console active and bring it forward when the corresponding event occurs.
Show console when a message is printed to standard error stream	The same as the previous option but for stderr.
Show in	Specify in which menus the command for running the tool should be included. Main menu means the Tools menu. The rest of the options correspond to context menus in various places.
Connection settings	In this section, appoint the to access through the SSH terminal and specify where the connection settings should be taken from: 1. Current Vagrant: select this option to have the commands in the SSH Terminal executed on the currently running Vagrant virtual machine. For details, see Vagrant . 2. Deployment server: select this option to have the commands in the SSH Terminal executed on the local or remote Web server accessible through one of the server access configurations . From the drop-down list, choose the server access configuration that specifies the destination environment and the settings to establish connection to it. – Select server on every run: if this option is selected, you will have to choose the desired server access configuration from the pop-up window, every time you choose Tools | Start SSH Session on the main menu. – If the desired server access configuration does not appear in the drop-down list, click the link Configure Servers, and define one in the Deployment page. For details, see Configuring Synchronization with a Web Server .
Tool settings	
Program	The path to the executable file to be run. Use to select the file, Insert macro to open the Macros dialog to select a macro. (Macros are resolved at runtime and let you specify context information such as currently selected file, your project source paths, etc.)
Parameters	The parameters to be passed to the program the way you'd specify them on the command line. Use Insert macro to open the Macros dialog to select a macro. When specifying the parameters, follow these rules: – Use spaces to separate individual parameters. – If a parameter includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If a parameter includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Working directory	The path to the current working directory for the program. Use to select the directory, Insert macro to open the Macros dialog to select a macro.

Server Certificates

On this page:

– Access

– Overview

– Table of controls

Access

File | Settings | Tools | Server Certificates for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Server Certificates for OS X

Ctrl+Alt+S

Overview

IntelliJ IDEA provides its own storage for trusted certificates. Use this page to manage this storage.

Table of controls

Item	Shortcut	Description
Accept non-trusted certificates automatically		Select this option if you want non-trusted certificates (i.e. the certificates that are not added to the list) to be accepted automatically, without sending a request to the server.
+	Alt+Insert	Add a trusted server certificate to the list. Select the certificate file in the dialog that opens . The certificate file should have an extension `.crt` , `.cer` or `.pem` . For a trusted certificate, the certificate information is shown in the lower part of the page.
–	Alt+Delete	Remove the selected trusted certificate from the list.

Settings Repository

File | Settings | Tools | Settings Repository for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Settings Repository for OS X

Ctrl+Alt+S

This page appears in the Settings/Preferences dialog, when Settings Repository plugin is enabled.

The plugin is bundled with IntelliJ IDEA and activated by default. If it is disabled, you can manually enable the plugin .

Use this page to configure the Settings Repository feature that allows you to share your IDE settings between different instances of IntelliJ IDEA (or other IntelliJ platform-based) products installed on different computers.

Item	Description
Auto Sync	Select this check box, if you want your local settings to be automatically synchronized with the settings stored in the repository every time you perform an Update Project or a Push operation, or when you close your project or exit IntelliJ IDEA. If this option is disabled, you can manually update your settings by choosing VCS | Sync Settings from the main menu.
Read-only sources	Use this section to configure additional repositories containing any types of settings you want to share, including live templates, file templates, schemes, deployment options, etc. These repositories cannot be overwritten or merged, just used as a source of settings as is. Use the following controls to manage the read-only repositories: + Click this button to add the URL of the GitHub repository that contains the settings you want to share. − Click this button to remove the selected repository from the list. Click this button to edit the URL of the selected source. Use these buttons to move up/down in the list. Click this button to clone the selected URL.

Startup Tasks

On this page, create a list of run/debug configurations to be launched automatically on the project start. This may be helpful if you run some Grunt or Gulp.js tasks or npm scripts on a regular basis. All you need is just add the run/debug configurations that launch such tasks or scripts to the list of startup tasks.

On this page:

– Access

– Table of Controls

Access

File | Settings | Tools | Startup Tasks for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Startup Tasks for OS X

`Ctrl+Alt+S`

Table of Controls

ItemTooltipDescription		
Run Configuration		This read-only field shows the names of the run configurations to be launched on the project start.
Shared		When this check box is selected, the corresponding task is available for other team members. Note that you can share only those run configurations that are already marked as shared in their definitions. If the directory-based project format is used, the settings for a run/debug configuration are stored in a separate .xml file in the `.idea\runConfigurations` folder if the run/debug configuration is shared, or in the `.idea\workspace.xml` file otherwise. If the file-based format is used, the settings are stored in the `.ipr` file for shared configurations, or in the `.iws` file otherwise.
+	Add	Click this button to add one of the run/debug configurations that are currently defined in the project to the list of tasks to be executed on the project start. Choose the relevant configurations from the list or choose Edit Configurations to open the Run/Debug Configurations dialog and define a required configuration in it.
–	Remove	Click this button to delete the selected task from the list.
	Edit	Click this button to open the Run/Debug Configurations dialog with the settings from the selected configuration and update them as required.

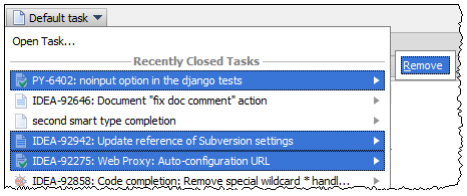
Tasks

File | Settings | Tools | Tasks for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Tasks for OS X

`Ctrl+Alt+S`

Use this page to set up the general options for tasks and context management .

Item	Description
Changelist name format	Type here the format of a changelist to be created. The available placeholders are: `{id},{summary},{number}, {project}`
Task history length	Specify the number of tasks to be stored in history.
Connection timeout	Sets the maximum time for a connection to be established. A value of zero means the timeout is not used. The default setting is five seconds (5000).
Always display task combo in toolbar	If this check box is selected, the combo box of tasks appears in the main toolbar. Use this combo box to switch between tasks , and remove the unnecessary tasks:  If this check box is not selected, then the tasks combo box hides, when the only open task is the default one.
Save context on commit	If this check box is selected, a new task with the name of the committed changelist is created upon commit. The new task will be available in the tasks combo on the main toolbar.
Enable issue cache	– Select this check box to have IntelliJ IDEA synchronize with the issue tracking system in the background on a regular basis, No matter whether you actually request on information from your issue tracker or not, IntelliJ IDEA will connect to your issue tracking system according to the specified frequency and refresh the cached issues. The advantage of this approach is that when you need to switch to a task, the up-to-date information is already at your disposal so you do not need to wait till IntelliJ IDEA establishes connection with the tracker and retrieves the information. Tip This configuration is especially recommended when working with rather "slow" issue tracking systems. – Clear this check box to have IntelliJ IDEA connect to the issue tracking system only when you actually need information on issues, clear the Enable issue cache check box.
Update (N) issues each (M) minutes	Specify how often IntelliJ IDEA should synchronize with your issue tracking system and refresh the cached issues.

Servers

File | Settings | Tools | Tasks | Servers for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Tasks | Servers for OS X

`Ctrl+Alt+S`

Use this page to associate your account in the bug tracking system you use with the IntelliJ IDEA project. The page is divided into the following sections:

- Configured servers
- Server options
  - General tab
  - Commit Message tab
  - Server Configuration tab

Configured servers

Item	Shortcut	Description
+	`Alt+Insert`	Click to add a new server of one of the supported types. For example: – YouTrack . Make sure that REST API in your YouTrack server is enabled. – JIRA – GitHub – Generic Click + to explore the list of the supported servers.
–	`Alt+Delete`	Click to remove the selected server.

Server options

General tab

Item	Available for	Description
Share URL	All servers	If this check box is selected, the server URL of your issue tracking system, specified in the field below becomes available to the other team members.
Server URL	All servers	Specify here the server URL of your issue tracking system. Tip Important note for YouTrack users: If you are using free instance of YouTrack InCloud, make sure to specify `http://<sitename>/youtrack` . If you are using a paid instance, specify `https://<sitename>/youtrack` .
Username / Password	All except for Lighthouse, Trello, GitLab, GitHub	Specify here login name and password for your account. Tip Important note for Sprintly users: The Password required here is the API Key from your Sprintly profile (not your Sprintly password). Refer to the Sprintly documentation for details.
Repository	GitHub	Specify here the name of the desired Git repository.
Search	YouTrack	Specify here the default search string.
Project ID	Lighthouse, Pivotal Tracker, Redmine	Specify here the name of your project in the tracker.
API Token	Lighthouse, Pivotal Tracker	Specify here the desired API token for the project in question.
Use proxy	All servers	Select this check box to use the specified proxy server.
Proxy settings	All servers	Click this button to set up proxy settings .
Use HTTP authentication	JIRA	Select this check box to enable HTTP authentication.
Test	All servers	Click this button to make sure the connection is established.
<Template variables>		More fields may be added to this tab depending upon the selected or cleared check box Show on first tab in the Template Variables dialog in the Server Configuration tab.

Commit Message tab

Item	Description
Add commit message	If this check box is selected, a commit message for a changelist, created from an issue in a bug tracking system, is formed according to the syntax defined below.
Message	Use the placeholders to define the commit message syntax. The available placeholders are `{id}` , `{summary}` , `{number}` , `{project}` .

Server Configuration tab

This tab only shows for the following servers:

- Generic
- Asana

– Assembla
– Sprint.ly

ItemDescription	
Login URL	Specify here the resource for authentication (if any). The request to this resource will take place every time the issues are retrieved from server. This field is disabled, if the check box Use HTTP authentication in the General tab is selected. The resource consists of a URL with possible template variables: – `{serverUrl}` : URL of the server, specified in the General tab. – `{password}, {username}` : correspond to the settings specified in the General tab. – `{max}` : maximum number of issues to be retrieved from the server. Also it contains the combo-box with the request type: GET or POST.
Tasks list URL	This resource enables obtaining the list of issues from the server and is requested every time the user opens tasks. The resource consists of a URL with possible template variables: – `{serverUrl}` : URL of the server, specified in the General tab. – `{password}, {username}` : correspond to the settings specified in the General tab. – `{max}` : maximum number of issues to be retrieved from the server. Also it contains the combo-box with the request type: GET or POST.
Single task URL	This resource makes it possible to obtain a more detailed information about a specific issue, using its ID. This field is mandatory, when the check box Each task in separate request is selected. The resource consists of a URL with possible template variables: – `{serverUrl}` : URL of the server, specified in the General tab. – `{password}, {username}` : correspond to the settings specified in the General tab. – `{max}` : maximum number of issues to be retrieved from the server. Also it contains the combo-box with the request type: GET or POST.
Response type	This section contains three radio buttons: – XML: click this radio button to return the server response in XML format. So doing, the selectors are described in the format XPath . – JSON: click this radio button to return the server response in JSON format. So doing, the selectors are described in the format JSONPath . – Text: click this radio button to retrieve issues fields using regular expressions.
Each task in separate request	Select this check box to make the field Single task URL mandatory.
Table of selectors	These selectors help specifying how exactly the detailed issue information will be extracted from the server response. The selectors have the following meanings: – tasks: path in the server response to the descriptions of specific issues (mandatory selector). – id: path to the unique ID within the description of a specific issue (mandatory selector). – summary: path to the title/header of an issue inside its description (mandatory selector). – description: path to the detailed description of an issue (optional selector). – updated/created: path to the time of updating or creating the issue respectively, in the format ISO-8601, inside its description (optional selector). – closed: path to a flag that defines whether a specific issue is opened or closed (optional selector). – issueUrl: path to a URL inside the issue description. It is possible to open information about this issue in your browser . Depending on the choice of response type , these selectors are described in XPath or JSONPath format, or as regular expressions.
Reset to Defaults	Click this button to discard all settings and return to the default values.
Manage Template Variables...	Click this button to show the dialog box Template Variables with the list of existing template variables. Use + and — buttons to manage the list. Note the check boxes in the columns Show on first tab and Hide. – If a check box in the column Show on first tab next to a custom template variable is selected, this custom variable is displayed in the General tab. – If a check box in the column Hide next to a custom template variable is selected, the value of this variable is shown dots. Otherwise, it is shown as is. Note that default variables cannot be deleted or changed!
Test	Click this button to make sure the settings are correct.

Time Tracking

File | Settings | Tools | Tasks | Time Tracking for Windows and Linux

IntelliJ IDEA | Preferences | Tools | Tasks | Time Tracking for OS X

Ctrl+Alt+S

Use this page to enable time tracking and configure its settings.

Item	Description
Enable Time Tracking	Select this check box to enable time tracking in the current project. When this check box is selected, the Time Tracking tool window becomes available on the right tool window bar, and on the View | Tool Windows menu.
Suspend delay	In this field, enter the "accuracy" of time tracking. If no user activity detected during this time, task is considered suspended.

Vagrant

File | Settings | Vagrant for Windows and Linux

IntelliJ IDEA | Preferences | Vagrant for OS X

Ctrl+Alt+S

Use this page to:

- Enable Vagrant support in IntelliJ IDEA by specifying the Vagrant executable file.
- Specify the location of the VagrantFile that determines creation of virtual boxes (instances) by provisioning predefined Vagrant base boxes through the `vagrant up` command.
- Handle the list of Vagrant base boxes to use in creation of virtual boxes (instances).

Before you start working with Vagrant, make sure that:

1. Vagrant is downloaded and installed.
2. Install and enable the Vagrant plugin as described in the sections Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .

Item	Description
Vagrant executable	Specify the fully qualified address of the executable file: `vagrant.bat` for Windows, `vagrant` for Unix and OS X. Type the path manually, or click the browse button and locate the desired file in the Select vagrant executable dialog box.
Instance folder	Specify here the fully qualified path to the directory, where the task `vagrant init` has been executed, and the `VagrantFile` is initialized and stored. A `VagrantFile` is a configuration file that defines the instance (virtual machine) you need. The file contains the virtual IP address, port mappings, and the memory size to assign. The file can specify which folders are shared and which third-party software should be installed. According to the `VagrantFile` your instance (virtual machine) is configured, provisioned against the relevant Vagrant base box, and deployed on your computer. A `VagrantFile` is created through the `vagrant init` command. When creation of an instance (virtual machine) is invoked either through the `vagrant up` command or through the Tools | Vagrant | Up menu option, IntelliJ IDEA looks for the `VagrantFile` in the directory specified in the Instance folder field. For more information, see http://docs.vagrantup.com/v2/vagrantfile/ . You can create a `VagrantFile` in any directory and appoint it as instance folder. If the field is empty, IntelliJ IDEA will treat the project root as the instance folder and look for a `VagrantFile` in it.
Provider	Use this field to specify the provider to be used by `vagrant up` command. If this field is left blank, the default provider is used.
Environment variable	Click the ellipsis button or press Shift+Enter to specify the shell variables to be used to configure the providers' behavior.
Boxes and Plugins tabs	
Boxes	This list shows the predefined Vagrant base boxes available in IntelliJ IDEA. Each item presents a Vagrant base box on which Vagrant configures and launches its instances (virtual machines). The entries of this list correspond to the output of the command `vagrant box list` .
+ Alt+Insert	Click this button to download a new base box. This command corresponds to `vagrant box add <name> <URL>` . By default, IntelliJ IDEA suggests the URL to the lucid32 box
− Alt+Delete	Click this button to remove the selected Vagrant base box. So doing, the box and the nested files are physically deleted from the disk. This command corresponds to `vagrant box remove <name>`
Plugins	Use this table to view and change the list of available plugins.
+ Alt+Insert	Click this button to install a new Vagrant plugin.
− Alt+Delete	Click this button to remove the selected plugin.
	Click this button to update the selected plugin.
	Use this button to attach a license to the selected plugin.

XPath Viewer

File | Settings | Tools | XPath Viewer for Windows and Linux

IntelliJ IDEA | Preferences | Tools | XPath Viewer for OS X

Ctrl+Alt+S

This page appears if XPath View+XSLT Support plugin is enabled. In this page, configure the IntelliJ IDEA behaviour during interactive execution of XPath expressions.

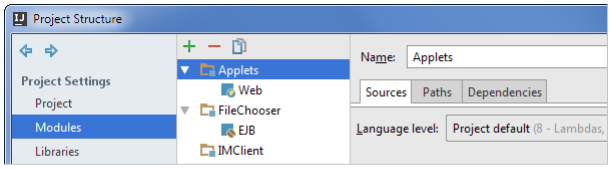
Item	Description
Scroll first hit into visible area	Select this check box to have the editor automatically scroll to the first XPath match.
Use node at cursor as context node	Select this check box to have the entered XPath expression use the currently selected node (tag/attribute/pi, etc.) as its context node and evaluate the expression relatively to this node.
Highlight only start tag instead of whole tag content	Do one of the following: – Select this check box to have only the name of a matching tag highlighted. – Clear this check box to have the entire content of a matching tag highlighted.
Add error stripe markers for each result	Select this check box to have each match supplied with an error stripe marker which can be quickly navigated to. The tooltip of each marker shows the matched content.
Show actions in Toolbar	Select this check box to have buttons that invoke XPath-related actions displayed on the Main Toolbar .
Show actions in Main Menu	When this check box is selected, XPath-related actions are available from the main menu.
Colors	In this area, configure color indication during execution of XPath expressions. Clicking on the color box will open the Select Color dialog in which you can modify the current color indication. – Highlight Color - in this area, select the color to indicate XPath matches in the editor. – Context Node Color - in this area, select the color to indicate the current context node.

Project Structure Dialog

File | Project Structure

`Ctrl+Shift+Alt+S`

The Project Structure dialog lets you manage your project and IDE-level elements such as modules , facets , libraries , artifacts and SDKs .



In most of the cases, there are two panes in the left-hand part that implement a two-level selector. The only exception is when you select Project. There is only one selector pane in this case.

On this page:

– Category selector
– Element selector

In this section:

– Project Page
– Modules
– Libraries and Global Libraries
– Facets
– Artifacts
– SDKs

Category selector

The leftmost pane is for selecting a category. There are two groups of categories in this pane.

The first group (Project Settings) provides access to the elements of your current project. In the second group (Platform Settings) are the IDE-level entities that are available in all of your projects.

When you select a category in the leftmost pane, the pane to the right shows a list (or, if appropriate, a tree) of elements belonging to this category.

The toolbar icons Back and Forward are for moving back and forward in the history of selected categories.

Element selector

The element selector pane lets you manage the elements in the corresponding category.

When you select an element, its settings are shown on a page to the right.

To perform various operations with the elements, use the toolbar icons, context menu commands (accessed by right-clicking an element), or keyboard shortcuts.

The following functions are available for all element types. Complete lists of commands for different element categories (modules, libraries, etc.) are provided in the corresponding sections.

Icon	Command	Shortcut	Description
+	New	`Alt+Insert`	Use this icon, command or shortcut to create a new element (module, library, etc.).
–	Delete	`Delete`	Use this icon, command or shortcut to delete the selected elements.

Project Page

File | Project Structure | Project

Ctrl+Shift+Alt+S | Project

| Project

Use this page to specify the project name, SDK , language level, and the compiler output path.

Item	Description
Project name	Use this field to edit the project name.
Project SDK	Select the project SDK . If the desired SDK is not present in the list, click New and select the necessary SDK type . Then, in the dialog that opens , select the SDK home directory and click OK. To view or edit the name and contents of the selected SDK, click Edit. (The SDK page will open.)
Project language level	Select the Java language level to be supported. The selected level will be used as the project default. The available options correspond to JDK versions: – 1.3 - Plain old Java – 1.4 - 'assert' keyword – 5.0 - 'enum' keyword, autoboxing, etc. – 6.0 - @Override in interfaces – 7.0 - Diamonds, ARM, multi-catch, etc. – 8.0 - Lambda support, type annotations, etc. An individual language level may be set for any of your modules . Note that if the target level is not explicitly defined (the compiler option `-target`) , it is considered equal to the source language level. Thus, if `-target` is not explicitly defined, it will be synchronized with the language level.
Project compiler output	Specify the path to the directory in which IntelliJ IDEA will store the compilation results. Click … to select the directory in the Select Path dialog . Two subdirectories in the specified directory will be created: – `production` for production code. – `test` for test sources. In these subdirectories, individual output directories will be created for each of your modules. The output paths may be redefined at the module level .

Modules

File | Project Structure | Modules

`Ctrl+Shift+Alt+S`

When you select the Modules category in the Project Structure dialog , a hierarchical view of existing module groups , modules , facets and, for Flash modules, build configurations is shown in the element selector pane . (Facets and build configurations are shown as module elements.)

Use the toolbar icons, context menu commands or keyboard shortcuts to manage the sets of elements shown (see below).

To view or edit the settings for an element (module, facet or build configuration), select the element of interest, and use the page to the right of the selector pane.

In this section:

- Module Page
- Facet Page
- Module Page for a Flash Module
- Build Configuration Page for a Flash Module

Toolbar icons, context menu commands and shortcuts

Icon	Command	Shortcut	Description
	New	`Alt+Insert`	Use this icon, command or shortcut to create a new module, facet, or, for a Flash module, build configuration.
	Delete	`Delete`	Use this icon, command or shortcut to delete the selected element (module, facet, etc.).
	Copy		Use this icon or command to create a copy of the selected module, or, for a Flash module, build configuration.
	Find Usages	`Alt+F7`	Use this icon, command or shortcut to find the usages of the selected module, facet or build configuration in the project.
	Hide or Show Module Groups		If there are module groups: use this icon or command to hide or show the module groups. See Grouping Modules .
	Expand All	`Ctrl+NumPad Plus`	Use this icon, command or shortcut to expand all the tree nodes to see their contents.
	Collapse All	`Ctrl+NumPad -`	Use this icon, command or shortcut to collapse all the tree nodes and show only the top-level nodes.

Module Page

File | Project Structure | Modules

Ctrl+Shift+Alt+S

The Module page opens in the right-hand part of the Project Structure dialog when you select a module in the element selector pane .

Use the Name field to edit the module name. Other settings are available on the following tabs:

- Sources Tab
- Paths Tab
- Dependencies Tab
- Plugin Deployment Tab
- Mobile Module Settings Tab
- Mobile Build Settings Tab

Sources Tab

Use the Sources tab of the Module page to select the supported language level for Java and to configure the module contents.

The module contents are configured by adding and removing the module content roots as well as by assigning individual folders (within the content roots) to categories such as sources and test sources, and also by excluding the folders.

– Language level list
– The left-hand pane
– The right-hand pane

Language level list

Item	Description
Language level	Use this list to select the Java language level for the module. The available options correspond to JDK versions. You can select the level set for the project (the option <Use project language level>) or set an individual level for the module.

The left-hand pane

The left-hand pane shows a tree of folders for a module content root . If the module has more than one content root, the structure shown corresponds to the content root selected in the right-hand pane.

The folders belonging to different categories have different icons.

The following table lists the available toolbar buttons (the Icon column) and explains their functions (the Description column). Note that the corresponding functions can also be accessed as the context menu commands. These are listed in the Command column.

Most of the icons/commands work as toggles, and can be used to cancel the corresponding assignment (to make a folder "an ordinary folder").

Icon	Command	Description
	Sources	Use this icon or command to assign the selected folder or folders to the source folder category.
	Tests	Use this icon or command to assign the selected folder or folders to test sources.
	Resources	For Java modules: use this icon or command to assign the selected folder or folders to resources.
	Test Resources	For Java modules: use this icon or command to assign the selected folder or folders to test resources.
	Excluded	Use this icon or command to make the selected folder excluded.
	New Folder	Use this command to create a folder in the selected folder. Specify the name for the new folder in the New Folder dialog that opens.

The right-hand pane

The right-hand pane shows the module content roots .

For each content root, a categorized view of the module folders is provided. The categories are the source folders, test source folders, etc.

The "ordinary" folders are not shown in this view.

The individual folders within the categories are identified by their paths. The folder paths are all relative to the module root folder (content root).

The folder paths, functionally, are hyperlinks that let you jump to the corresponding folders in the tree shown in the left-hand pane.

If a module has more than one content root, selecting a content root in the right-hand pane also switches the tree view in the left-hand pane. That is, when you click somewhere within the content root area, the folder structure of this particular content root is shown in the left-hand pane.

The following table lists the controls available in the right-hand pane (icons) and describes their functions.

Icon	Tooltip	Description
	Add Content Root	Use this icon to add a content root. Select the folder to be added as a content root in the dialog that opens .
	Remove Content Entry	Use this icon to remove the corresponding content root from the list of the content roots.
	Edit properties	Available in Java modules for folders marked as containing sources or resources. Use this icon to open the Edit Root Properties dialog in which you can specify: – A package prefix for the selected source folder. Specifying the package prefix (e.g. `com.mycompany.myapp`) eliminates the necessity to create the corresponding folder structure (e.g. `com/mycompany/myapp`). For more information, see Package prefix for Java source roots. See also, Assigning a package prefix to a folder with Java sources. – An output path for the selected resource folder. See Specifying the output path for resources. – Whether the folder contains generated sources or resources (the For generated sources or For generated resources check box). See Specifying that the sources or resources are generated .
	Unmark	Use this icon to remove the folder from the corresponding category. As a result, the folder becomes "an ordinary folder", that is, not belonging to any specific category.

Paths Tab

Use the Paths tab of the Module page to configure the compiler output paths for the module, and also to specify the locations of external JavaDocs and external annotations associated with the module.

- Compiler output
- JavaDoc
- External Annotations

Compiler output

Item	Description
Inherit project compile output path	Select this option to use the paths specified for the project .
Use module compile output path	Select this option if you want to set the compiler output paths for this module individually. Specify the associated settings: – Output path. Specify the directory for the production classes. Use ... (Shift+Enter) to select the directory in the Select Output Path dialog . – Test output path. Specify the directory for the test classes. Use ... (Shift+Enter) to select the directory in the Select Test Output Path dialog . – Exclude output paths. Select this check box to make the output directories excluded .

JavaDoc

Use the available controls to compose the list of locations where external JavaDocs associated with the module are stored.

Icon	Shortcut	Description
+	Alt+Insert	Use this icon or shortcut to add a JavaDoc file to the list.
–	Alt+Delete	Use this icon or shortcut to remove the selected item from the list.
		Use this icon to add a URL of online JavaDoc.

External Annotations

Use + and – to manage the list of locations (directories) for external annotations associated with the module.

Icon	Shortcut	Description
+	Alt+Insert	Use this icon or shortcut to add a directory to the list.
–	Alt+Delete	Use this icon or shortcut to remove the selected item from the list.

Dependencies Tab

On this tab, you can define the module SDK and form the list of module dependencies .

- Main settings and controls
- Context menu commands for dependency items
- Sorting the list of dependencies

Main settings and controls

Item	Description
Module SDK	Select the module SDK . (To associate the project SDK with the module, select Project SDK. Note that if you change the project SDK later, the module SDK will change accordingly.) If the desired SDK is not present in the list, click New and select the necessary SDK type . Then, in the dialog that opens , select the SDK home directory and click OK. To view or edit the name and contents of the selected SDK, click Edit. (The SDK page will open.)
List of dependencies	Shown below the module SDK is the list of module dependencies . Use +, −, ↑ and ↓ to add, remove and reorder the items (libraries and modules). Use the cells in the header row to sort the list, see Sorting the list of dependencies. When you compile or run your code, the list of dependencies is used to form the classpath for the compiler or JVM. (Native Library Locations, if any, are added to `java.library.path` rather than the classpath.) Export. This option lets you control the compilation classpath for the modules that depend on this one. Select the check box if you want the item to be exported as a dependency along with the module. That is, if there is a module that depends on this one, the items with the Export option on will be included in the compilation classpath of the dependent module. Note that this setting doesn't affect the runtime classpath. At runtime, all the dependencies of the current module are included in the classpath for the modules that depend on this module. Scope. This setting lets you control the classpath separately for your sources and test sources , and for the build and the run phases. (The classpath may be different when 1) your sources are compiled 2) your test sources are compiled 3) your compiled sources are run 4) your tests are run. The Scope option defines the classpaths in which the dependency is to be included.) Select the necessary option from the list: – Compile. The dependency is included in the classpath for your sources and test sources at the compilation and run phases. – Test. The dependency is included in the classpath only for your test sources at the compilation and run phases. – Runtime. The dependency is included in the classpath for your sources and test sources but only at the run phase. – Provided. For your sources, the dependency is included in the classpath only at the compilation phase. This is useful when there is a container (e.g. a web container of an application server) that provides the corresponding dependency at runtime. For your test sources, the dependency is included in the classpath both at the build and run phases. Application server libraries , normally, are included in dependency lists with the scope Provided. Note that IntelliJ IDEA is a bit different from some other build tools (e.g. Gradle and Maven) in the way it processes dependencies for test sources. If your module (say, module A) depends on another module (module B), IntelliJ IDEA assumes that the test sources in module A depend not only on the sources in module B but also on its test sources. Consequently, the test sources of module B are also included in corresponding classpaths. + (`Alt+Insert`). Use this icon or shortcut to add a library or a module to the list of the module dependencies. Select one of the following options: – Jars or directories. Select this option to create a new module library and add it to the list of dependencies. In the dialog that opens , select the files and folders to be included in the library. These may be individual `.class` and `.java` files, directories and archives (`.jar` and `.zip`) containing such files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`). – Library. Select this option to add one or more of the existing project, global or application server libraries to the list of dependencies. In the dialog that opens, select the library or libraries and click Add Selected. – Module Dependency. Select this option to specify the modules that the current module should depend on. − (`Alt+Delete`). Use this icon or shortcut to remove the selected item or items from the list of dependencies. ↑ (`Alt+Up`) and ↓ (`Alt+Down`). Use these icons and shortcuts to move the selected item up or down in the list. See Order of dependencies. (`F4`). Use this icon or shortcut to open the Configure Library dialog to edit the library.
Dependency storage format	Select the format for storing the dependencies (as an IntelliJ IDEA module, or as Eclipse project). This option is helpful for the teams that use different development tools.

Context menu commands for dependency items

Item	Description
Edit	For a library: use this command to edit the selected library. (The Configure Library dialog will open.)
Remove	Use this command to remove the selected item or items from the list of dependencies.
Navigate (`F4`)	Use this command or shortcut to see details for the selected item. An appropriate page of the Project Structure dialog will open, if any.
Find Usages	Use this command to see the list of modules where the selected dependency is used. Clicking a module in that list, opens the Module Page for the corresponding module.
Analyze This Dependency	Use this command to perform a dependency analysis for the selected item and display the results in the Dependency Viewer .
Move to Project Libraries or Move to Global Libraries	For a module library: Use this command to move the selected library to a higher level (project or global).
Copy to Module Libraries	For a global or project library: Use this command to create a copy of the selected library at the module level .

Sorting the list of dependencies

You can sort the dependencies by their names and scopes by clicking the cells in the header row.

If you click a cell once, the list is sorted by the corresponding column in the ascending order. The sorting marker appears in the cell: ▲. When you click the cell for the second

time, the information is sorted in the descending order. To show this, the sorting marker changes its appearance: . Finally, when you click the cell for the third time, the initial unsorted state is resorted.

Note that the sorting operations don't change the actual order of dependencies.

When the list is sorted, the icons for changing the order of dependencies are inactive.

Plugin Deployment Tab

Use the Plugin Deployment tab of the Module page to specify the settings related to deploying your plugin.

Note that this tab is available for Plugin modules only, see Plugin Development Guidelines .

Item	Description
Path to META-INF\plugin.xml	Specify the path to the directory in which the directory `META-INF` with the file `plugin.xml` inside should be located. Use … (Shift+Enter) to select the directory in the Select META-INF Directory Location dialog . `plugin.xml` is a plugin descriptor that contains general information about the plugin. This information includes the plugin name, description and version, the lowest IntelliJ IDEA build number with which it works as well as descriptions of the component and actions. IntelliJ IDEA needs this file to be able to load the plugin. `plugin.xml` should be located in the directory with the name `META-INF` .
Use user manifest	Select this check box if you want a custom manifest file to be included in the plugin distribution. Specify the path to the file. Use … (Shift+Enter) to select the file in the Select manifest.mf dialog .

Mobile Module Settings Tab

Note that this tab is available for J2ME modules only. The settings depend on the Java Mobile toolkit (SDK) being used, see J2ME .

- Settings for Java Wireless Toolkit (WTK)
- Settings for DoJa

Settings for Java Wireless Toolkit (WTK)

Item	Description
MIDlet-Name	In this text box, specify the MIDlet suite name (corresponds to the JAD MIDlet-Name property). This option is necessary to identify MIDlet suite on a device.
MIDlet-JAR-URL	In this text box, specify the MIDlet JAR location. This option is also necessary to identify MIDlet suite on a device. The JAR file will be installed from the location MIDlet-JAR-URL afterward. You can specify the location manually, or click the ellipsis button and select the necessary location in the dialog that opens .
MIDlet-Vendor	In this text box, specify the MIDlet vendor name, that is, the MIDlet suite provider.
MIDlet-Version	In this text box, specify your MIDlet version number.
Optional Settings	Click this button to open the Optional MIDP Settings dialog box where you can set additional options that are not generally required.
Defined MIDlets	In this area, define the MIDlets to form your MIDlet suite. A defined MIDlet contains the following information: – The MIDlet name under which the MIDlet will be presented to the user. – The icon to indicate the MIDlet (optional). – The name of the class that implements the MIDlet and that will be called by the application manager to load the MIDlet. Use the buttons Add, Edit and Remove to create, update, and delete MIDlet definitions. Use the buttons Move Up and Move Down to change the order of MIDlets. This order determines the order in which the MIDlets are invoked.
Keep user defined JAD file	Select this check box to create the JAD file with the settings specified above.

Settings for DoJa

Item	Description
AppName	In this text box, type the application name (50 bytes maximum).
PackageUrl	In this text box, specify the URL address to access the application. Make sure the URL is an ASCII-format string. Also, an IP address cannot be specified directly.
AppClass	In this text box, specify the application main class name (255 bytes maximum). This should be a subclass name of `com.nttdocomo.ui.IApplication` .
Optional Settings	Click this button to open the Optional MIDP Settings dialog box where you can set additional options that are not generally required.
Keep user defined JAM file	Select this check box to create the `JAM` file with settings specified above.

Optional MIDP Settings Dialog

The dialog box opens when you click the Optional Settings button on Mobile SDK Specific Options page of the New Project From Scratch Wizard.

Use this dialog box to specify additional options that are not generally required. The contents of the dialog box depend on the mobile development tool used:

– Java Wireless Toolkit (WTK) .

– DoJa .

WTK

Item	Description
MIDlet-Description	In this text box, provide a description of your MIDlet suite.
MIDlet-Icon	In this text box, specify the location of the MIDlet suite presentation `.png` file within the `.jar` .
MIDlet-Data-Size	In this text box, type the minimum number of bytes of persistent data required by the MIDlet.
MIDlet-Info-URL	In this text box, specify the URL to access more information about the MIDlet suite and/or the vendor.
MIDlet-Delete-Confirm	In this text box, type the message to be shown when the user is prompted to confirm deletion of the MIDlet suite.
MIDlet-Install-Notify	In this text box, specify the URL to send `POST` request to confirm successful installation of this MIDlet suite.
User Defined Settings	In this area, compose a list of user-defined attributes related to specific MIDlets. For each attribute, specify a key and a value. Use the Add and Remove buttons to manage the contents of the list. Use the Move Up and Move Up buttons to define the order of settings. This order determines the priority in which the settings are applied.

DoJa

Item	Description
AppVersion	In this text box, specify the application version (10 bytes maximum).
ConfigurationVer	In this text box, specify the J2ME configuration version, for example, `CLDC-1.0` .
ProfileVer	In this text box, specify the version of the i-mode Java Application runtime environment profile, for example, `DoJa-1.5oe` .
SPsize	In this text box, specify the size of the ScratchPad in bytes.
AppParam	In this text box, specify the parameters of the main class (255 bytes maximum).
UseNetwork	In this text box, specify `http` for applications that use network functionality.
TargetDevice	Specify here a model name if the application is targeted to a particular model(128 bytes maximum). Do not set this option for applications targeted to all models.
LaunchAt	To have your application launched automatically at the specific time, specify the required time in this text box.
AppTrace	Choose the On value to have some information output using `System.out.println()` or `System.err.println()` after the i-Appli is terminated.
DrawArea	In this text box, specify the size of the application drawing area, for example, `120x130` .
GetUtn	When this option is selected, the application refers to the handset identification code and IC chip information on its SIM or UIM card.

Mobile Build Settings Tab

Note that this tab is available for J2ME modules only.

Item	Description
JAR File	In this text box, specify the location of the `JAR` file that will contain all the MIDlet classes in the suite, the Java classes shared between MIDlets, and the resource files.
JAD/JAM File	In this text box, specify the location of the `JAD` or `JAM` file (depending on the SDK used). This file contains a predefined set of attributes (such as `MIDlet-Name`, `MIDlet-JAR-URL`, etc.) according to which the device application management software identifies, retrieves, and installs your application.
Use user manifest	Select this check box to have the `JAR` file supplied with a custom manifest file. If this case, the `MIDlet-Name`, `MIDlet-Version`, and `MIDlet-Vendor` should have the same values in both the `JAD` and the manifest files, otherwise the application manager will fail to load the `JAR` file.
Setup mobile exploded directory	Select this check box, to have a copy of the mobile module package as a separate folder and specify the location of this folder in the text box below. Tip For WTK, you can set the exploded directory as the directory where the emulator takes applications. This means that it should correspond to the `WTK_Installation_Dir/apps/Project_Name/bin/` directory.
Exclude from module content	Select this option to have IntelliJ IDEA exclude the replicated package from the module contents.
Create Mobile Resources Directory	Select this check box to have resources stored in a separate folder and specify the location of this folder in the text box below.

Facet Page

File | Project Structure | Modules

Ctrl+Shift+Alt+S

The Facet page opens in the right-hand part of the Project Structure dialog when you select a facet in the element selector pane .

Facet settings vary depending on the facet type:

- Android Facet Page
- AspectJ Facet Page
- Android-Gradle Facet Page
- Bean Validation Facet Page
- Google App Engine Facet Page
- GWT Facet Page
- Hibernate and JPA Facet Pages
- JSF Facet Page
- OSGi Facet Page
- Seam Facet Page
- Spring Facet Page
- Struts Facet Page
- Struts 2 Facet Page
- Tapestry Facet Page
- Web, EJB and Java EE Application Facet Pages
- Web Services Facet Page
- Web Services Client Facet Page

Android Facet Page

File | Project Structure | Modules - module - Android facet

`Ctrl+Shift+Alt+S`

Use this page to configure the settings of an Android facet attached to a specific module.

In this section:

– Common Android facet options
– Structure tab
– Generated Sources tab
– Packaging tab
– ProGuard tab
– Multi-dex tab

Common Android facet options

In this area, configure the facet general settings.

Item	Description
Library module	Select this check box to turn this module into a library module, so that other Android application projects can reference its source code and resources.
Update "project.properties" file automatically	Select this option if you want the `project.properties` file to be updated automatically when one of the following options is enabled or disabled: – Library module – Enable manifest merging
Reset paths to defaults	Click this button to return to the default Android facet settings.

Structure tab

In this tab, specify the location of the key application components in the module tree structure. Based on these settings, IntelliJ IDEA supports code completion, resolves references, and provides other types of coding assistance.

Item	Description
Manifest file	In this text box, specify the path to the AndroidManifest.xml file. This file contains the information that is required to run the application. It must be located in the module root directory.
Resources directory	In this text box, specify the path to the folder where the application resources are stored. Resources located in this directory are assigned IDs and can be referenced through the `R.java` file or from XML resource definition files. The default location is `<module root>/res`.
Assets directory	In this text box, specify the path to the folder where the application assets are stored. Files located in this directory are not assigned IDs and cannot be referenced through the `R.java` file or from XML resource definition files. You can access this directory like a normal file system and read data from these files using the AssetManager. The default location is `<module root>/assets`.
Native libs directory	In this text box, specify the path to the folder where the Android native libraries are stored. The default location is `<module root>/libs`.

Generated Sources tab

In this tab, specify the location of the application source files.

Item	Description
Generate sources automatically	Select this option if you want the `R.java`, `AndroidManifest.jave` and `.aidl` files to be generated automatically based on the definitions of resources and the `AndroidManifest.xml` file.
R.java and Manifest.java files	In this area, specify the target folder for the `R.java` file that contains IDs of all resources defined in your project, and the `AndroidManifest.jave` file that contains permissions. To change the default location, type the path manually or click the Browse button … and select the target folder in the dialog that opens.
AIDL files	In this area, specify the target folder for the .aidl files generated by the AIDL Compiler. To change the default location, type the path manually or click the Browse button … and select the target folder in the dialog that opens.

Packaging tab

In this tab, configure the behavior of the Android Asset Packaging Tool (aapt) that is responsible for creating an `.apk` file.

Item	Description
Use resource directory specified at "Structure" section	Select this option if you want the compiler to use the resources from the location specified in the Resources directory field in the Structure tab.
Use custom resource directory	Select this option if you want the package to contain resources from a location different from the one specified in the Resources directory field in the Structure tab. Type the path manually or click the Browse button … and select the target folder in the dialog that opens.
Include assets from dependencies into APK	Select this option if you want to include assets from dependencies into the application package.
Rename manifest package	Select this option if you want the application ID to be changed on build time, and specify the new name (for more information, see Renaming an Application Package).

	This option is only available for Application modules.
Enable manifest merging	Select this option if you want to automatically merge manifest files of library modules with the manifest file of the application that contains this library module (for more information, see Sharing Android Source Code and Resources Using Library Projects). This option is only available for Application modules.
Additional command line parameters	In this text box, type the additional parameters to be passed to the Android Asset Packaging Tool(aapt). If the set of additional parameters does not fit into the text box, click and specify the parameters in the dialog that opens. For example, if you want to include resources of a certain type in an uncompressed format, type `-0 <file extension for this type of resources>` .
APK path	Specify the target directory for the `.apk` file that will be generated as a result of Android module compilation. Select a folder from the drop-down list, or click the Browse button and specify the path in the dialog that opens.
Custom debug keystore	In this text box, specify the location of the keystore where the debug key you want to use is located. Type the path manually or click the Browse button and select a folder in the dialog that opens.
Include test code and resources into APK	Select this option to include sources and resources located under the test roots into the debug APK created on build time. Test data is never included in the release APK that is generated via the Generate Signed APK wizard .
Pre-dex external jars and Android library dependencies	During the application packaging, the `.class` files of a library module are converted into `.dex` files. This operation is referred to as dexing. Finally, the `.dex` files output from the library module is included in the final application `.apk` (learn more about the building procedure from Building and Running). As a rule, the contents of a library module remain unchanged. In this case you can have them `dexed` only once, whereupon the output `.dex` files are included in the `.apk` . This approach is referred to as pre-dexing. By default, IntelliJ IDEA pre-dexes library mode dependencies as well as external `jars` that have not been updated since the previous build. You can change these settings so that all `.class` files are always dexed. When this option is selected, `.dex` files output from `.class` files of library modules or external `.jars` are pre-dexed. That means they are not dexed anew if the corresponding `.class` files have not been updated since the previous build. If this check box is cleared, all `.class` files are dexed on each build. This option is unavailable for Library modules.

ProGuard tab

In this tab, enable the ProGuard tool used to obfuscate the application during packaging.

Item	Description
Proguard logs directory	This text box shows the default location of the ProGuard logs. To modify the location, click the Browse button and select a folder in the dialog that opens.
Run ProGuard when building debug APK	Select this option if you want IntelliJ IDEA to obfuscate the debug APK through integration with the built-in ProGuard tool. Note that for release application packages, there is a dedicated check box in the Generate Signed APK wizard and in Project Structure | Artifacts | Android tab .
Config file paths	This text box shows the default location of the `proguard-project.txt` configuration file that is created automatically together with an Android module. To modify the location, click the Browse button and select a folder in the dialog that opens.

Multi-dex tab

In this tab you can configure the multi-dex support .

Item	Description
Enable multi-dex support	Select this check box to enable Android multi-dex feature.
Main dex list	Specify the main dex list.
Minimal main dex	Select this check box for minimal main dex. This option lets you put only classes that are selected by main dex list into the main dex.

AspectJ Facet Page

File | Project Structure | Modules - module - AspectJ facet

Ctrl+Shift+Alt+S

The page is available only if the AspectJ support plugins are enabled. See Enabling AspectJ Support Plugins .

On this page, you can:

- Specify that the AspectJ compiler ajc should be used only for post-compile weaving.
- Specify the `ajc` command-line option `aspectpath` for the module.

Item	Description
Post-compile weave mode	If you select this check box, `javac` is used to compile the source code. Then, `ajc` is used to weave the compiled class files. As a result, the overall process takes less time. IMPORTANT: Don't select this check box if there are `.aj` aspects in the module. That is, you should select this check box only if all the aspects are defined as `@Aspect` -annotated Java classes (in `.java` files).
Aspect path	Use the available controls to form the `aspectpath` for the module. (Alt+Insert). Use this icon or shortcut to add libraries and other modules. Select the necessary libraries and modules in the dialog that opens. (To choose from, dependencies of the module are suggested.) (Alt+Delete). Use this icon or shortcut to remove the selected items from the list. (Alt+Up). Use this icon or shortcut to move the selected item one line up in the list. (Alt+Down). Use this icon or shortcut to move the selected item one line down in the list.

Android-Gradle Facet Page

File | Project Structure | Modules - module - Android-Gradle facet

`Ctrl+Shift+Alt+S`

Use this page to configure the settings of an Android-Gradle facet attached to a specific module.

In this section:

– Properties tab

– Signing tab

– Flavors tab

– Build Types tab

– Dependencies tab

Properties tab

This tab lets you specify properties for your `build.gradle` file.

Item	Description
Compile SDK Version	Use this drop-down list to select the compilation target.
Build Tool Version	Use this drop-down list to select a version of the build tool.
Library Repository	Specify the library repository. By default, `jcenter` repository is used for the Android-Gradle application. For more information, see jcenter repository .
Ignore Assets Pattern	Use this field to specify asset patterns that you want to ignore.
Incremental Dex	Use this drop-down list to set an incremental dexing. You can choose true or false option, or leave it blank.
Source Compatibility	Use this drop-down list to select the Java version compatibility for the Java source compilation.
Target Compatibility	Use this drop-down list to select the Java version to generate Java classes.

Signing tab

This tab lets you specify settings for signing configurations. The fields become editable when you add the `config` class in the left-hand area of the tab. You can also delete `config` classes that you don't need.

Item	Description
Name	This field displays a name of the `config` class that you have selected. You can modify the name, however, it should be a recognizable Java identifier.
Key Alias	Use this field to enter the name of your private key store file.
Key Password	Use this field to enter the password for the private key.
Store File	Use this field to enter a location of your private key store file. Alternatively, click Browse button … and select the target folder in the dialog that opens.
Store Password	Use this field to enter the password of your store file.

Flavors tab

This tab lets you specify settings for flavors' configuration. The fields become editable when you add the `flavor` in the left-hand area of the tab. You can also delete the flavor entry that you don't need.

Item	Description
Name	This field displays a name of the build flavor that you have selected. You can modify the name.
Min Sdk Version	Use this field to add the minimum SDK version for the flavor.
Application Id	Use this field to add an application ID.
Proguard File	Use this field to enter the location of the Proguard file . Alternatively, click Browse button … and select the target folder in the dialog that opens.
Signing Config	Use this drop-down list to select a name of the `config` class that you have selected in the Signing tab.
Target Sdk Version	Use this drop-down list to select the target SDK version for your build flavor.
Test Instrumentation Runner	Use this field to enter the name of a class for running test cases. By default, the test class name is android.test.InstrumentationTestRunner.
Test Application Id	Use this field to enter an application id for your testing.
Version Code	Use this field to enter an integer value that represents the version of the application code, relative to other versions.
Version Name	Use this field to enter a string value that represents the release version of the application code, as it should be shown to users.

Build Types tab

This tab lets you specify the settings for Android build types . By default, debug and release versions are created and displayed in the left-hand area of the tab. You can add a new entry or delete an existing one.

Item	Description
Name	This field displays a name of the build type that you have added. You can modify the name.
Debuggable	Use this property to set it to either `true` or `false` depending on the build type.
Jni Debug Build	Use this property to set it to either `true` or `false` depending on the build type.
Signing Config	Use this drop-down list to select a name of the `config` class that you have selected in the Signing tab.

Renderscript Debug Build	Use this property to set Renderscript debug build to either `true` or `false` depending on the build type.
Renderscript Optim Level	Use this property to set the optimization level for the renderscript compiler. The default value is 3.
Minify Enabled	Use this property to set the minification of your code to either `true` or `false` depending on the build type.
Pseudo Locales Enabled	Use this property to set the pseudo locales to either `true` or `false` depending on the build type.
Proguard File	Use this field to enter the location of the Proguard file. Alternatively, click Browse button [...] and select the target folder in the dialog that opens.
Application Id Suffix	Use this field to enter the suffix of your application id.
Version Name Suffix	Use this field to enter the version name suffix.
Zip Align	Use this drop-down list to set zipalign for the file. You can use either `true` or `false` depending on the build file.

Dependencies tab

Use this tab to specify dependencies for your application.

Item	Description
Scope	Use this setting to control the classpath separately for your sources and test sources, and for the build and the run phases. (The classpath may be different when 1) your sources are compiled 2) your test sources are compiled 3) your compiled sources are run 4) your tests are run. The Scope option defines classpaths in which you include the dependency.) You can select from the following options: – Compile - use this option to include the dependency in the classpath for your sources at the compilation and run phases. – Provided - use this option include a dependency that is provided by the runtime environment. – APK - use this option to include the dependency in .apk file that will be generated as a result of Android module compilation. – Test compile - use this option to include the dependency in the classpath for your test sources at the test compilation phase. – Debug compile - use this option to include the dependency in the classpath for the debug build type at the compilation phase. – Release compile - use this option to include the dependency in the classpath for the release build type at the compilation phase.
+ (`Alt+Insert`)	Use this icon or shortcut to add a library or a module to the list of the module dependencies. Select one of the following options:> – Jars or directories. Select this option to create a new module library and add it to the list of dependencies. In the dialog that opens, select the files and folders to be included in the library. These may be individual `.class` and `.java` files, directories and archives (`.jar` and `.zip`) containing such files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`). – Library. Select this option to add one or more of the existing project, global or application server libraries to the list of dependencies. In the dialog that opens, select the library or libraries and click Add Selected. – Module Dependency. Select this option to specify the modules that the current module should depend on.
– (`Alt+Delete`)	Use this icon or shortcut to remove the selected item or items from the list of dependencies.
↑ (`Alt+Up`) and ↓ (`Alt+Down`)	Use these icons and shortcuts to move the selected item up or down in the list. See Order of dependencies.

Bean Validation Facet Page

File | Project Structure | Modules - module - Bean Validation facet

Ctrl+Shift+Alt+S

The page is available only for modules with enabled Bean Validation support.

Use this page to configure the individual settings of a Bean Validation facet attached to a particular module.

Item	Description
Hibernate Validator	Select the version of the Hibernate Validator to be used.

Google App Engine Facet Page

File | Project Structure | Modules - module - Google App Engine facet

Use this page to view and change Google App Engine facet settings.

Note that the Google App Engine facet is not tied to the Web facet. Both facets are independent.

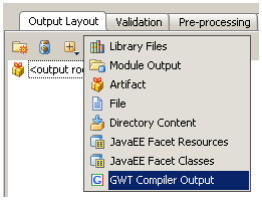
Item	Description
Path to Google App Engine SDK installation directory	Specify the path to the directory where Google App Engine SDK for Java is installed.
App Engine account	In this section, specify your Google account credentials.
Run Enhancer for the following classes and packages on make	If this check box is selected, the `datanucleusenhance` step will be executed after each make. Use Add and Remove buttons to specify the classes and packages to be processed by the enhancer. By default, the state of this check box corresponds to the facet settings made on creating the Google App Engine project .
Persistence	This field shows persistence type defined on creating the Google App Engine project . You can select a different one from the drop-down list. Selected persistence type defines which particular enhancer will be used on compilation.

GWT Facet Page

File | Project Structure | Modules - module - GWT facet

Ctrl+Shift+Alt+S

Use this page to configure the individual settings of a GWT facet attached to a particular module.

Item	Description
Path to GWT installation directory	In this drop-down list, select the location of the GWT SDK. If necessary, click ... and select the necessary location in the dialog that opens .
Target Web Facet	In this drop-down list, select where GWT components should be ascribed. The available options are: – None: the generated files will not be ascribed to any web facet. – Web: the generated files will be ascribed to a `war` /exploded directory. This option is available only if the GWT application is placed in a Web module with a Web facet .
JavaScript output style	In this drop-down list, select the optimization style that the compiler will apply to the generated JavaScript code. The following options are available: – Obfuscated. Choose this option to get the output smaller and thus faster to load. The function names in the output JavaScript code are unpredictable. – Pretty. Choose this option to get the output readable to a human. – Detailed. Choose this option to get the output with even more detail, for example, verbose variable names.
Compiler maximum heap size	In this text box, specify maximum heap size in Mb that will be used by the GWT compiler. In order to have GWT Compiler launched, you need to add GWT Compiler Output element to the artifact. (Project Structure | Artifacts | Output Layout)  Thus GWT Compiler is launched on artifact build, and the generated files are stored in the artifact's output directory.
Additional compiler VM options	In this text box, specify additional VM options, for example, the ones for compiling and running large GWT projects. When specifying the options, follow these rules: – Use spaces to separate individual options, for example, `-client -ea -Xmx1024m` . – If an option includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some" "arg` or `"some arg"` . – If an option includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `-Dmy.prop=\"quoted_value\"` .
Compiler parameters	In this text box, specify the options to be passed to the GWT compiler process in the format of a command line. Use the same rules as for specifying the VM options .
GWT Module	This read-only list shows all the GWT modules in your project. To have a GWT module involved in the make process , select the check box next to the name of the module in the list. The make process is automatic re-compilation of all the module or project sources that have been changed since the last compilation.
Output Relative Path	In this text box, specify the output path relative to the GWT modules compile output root.
Download GWT	Click this button to download the GWT SDK.
Create Artifact	To launch GWT compiler, you need to have an artifact configured with GWT Compiler output element added in the Output Layout tab. Click this button to automatically create an artifact with configured output layout.

Hibernate and JPA Facet Pages

File | Project Structure | Modules - module - JPA or Hibernate facet

Use this page to manage configuration and object/relational mapping files, and to download missing libraries.

Item	Description
Descriptors	Use the controls in this section to manage configuration and O/R mapping files such as `hibernate.cfg.xml` for Hibernate, and `persistence.xml` and `orm.xml` for JPA. + (`Alt+Insert`). Add an existing file or create a new one. In the dialog that opens, specify the file location and name. (`Enter`). Replace the selected file with another (existing) file. In the dialog that opens, specify the file that you want to use as a replacement. – (`Alt+Delete`). Remove the selected file from the list. In the dialog that opens, specify whether you want to physically delete the file. (Otherwise, only the file association with the facet will be removed.)
Default JPA Provider	For JPA: When `persistence.xml` is created, this setting affects the `<provider>` element in that file. For example, `<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>` will be generated for EclipseLink. There will be no `<provider>` element if <no provider> is selected.
Fix	If there are missing libraries, click this button to fix the problem. (The Setup Library Dialog will open.)

JSF Facet Page

File | Project Structure | Modules - module - JSF facet

Ctrl+Shift+Alt+S

Use this page to configure the JSF facet settings.

Note that you can download the JSF component libraries such as ICEfaces, OpenFaces, PrimeFaces and RichFaces by "adding" the corresponding facets to a JSF facet.

Item	Description
Facelets support	Select the required Facelets support option. This is the project-level setting. The selected option is set for all modules in the current project. – Auto. Automatic Facelets detection. The XHTML files are treated as Facelets or ordinary XHTML files depending on whether the module has a JSF facet or not. In the modules that have a JSF facet, all XHTML files are considered Facelets. If a module does not have a JSF facet, all XHTML files in it are treated as ordinary XHTML files. – Enabled. All XHTML files in the project are treated as Facelets. – Disabled. All XHTML files in the project are considered not to be Facelets and are treated as ordinary XHTML files.

OSGi Facet Page

File | Project Structure | Modules - module - OSGi facet

`Ctrl+Shift+Alt+S`

Use this page to configure support of OSGi bundles generation.

Note Prerequisite: Osmorc plugin should be enabled.

The available tabs are:

- General
- Bundle JAR
- Manifest Generation

General Tab

Item	Description
Bundle Creation	In this area, specify the bundling method and appoint the OSGi Bundle manifest file to use (or specify the way to create a new one). The available options are: – Create manifest using facet settings and bundle using facet configuration - select this option to have bundles assembled according to the configuration settings specified in the Bundle JAR tab and the manifest file generated according to the settings of the Manifest Generation tab. – Use existing manifest and bundle using facet configuration - select this option to have bundles assembled according to the configuration settings specified in the Bundle JAR tab and use an existing manifest file of your choice. In the Manifest file location area, appoint the manifest file to apply. – Select the Use project default option to use the manifest file that is appointed default for the current project in the OSGi: Project Settings dialog box. – Alternatively, select Custom option to use a custom manifest file. In the text box, type the path to the desired file relative to the project root or click the Browse button … and choose the desired location in the dialog that opens . – Create using bnd and ignore facet configuration - select this option to have bundles assembled using the bnd. In the Bnd file location text box, specify the path to the `.bnd` file to use. Type the path manually or click the Browse button … and choose the desired location in the dialog that opens . – Create using Bundlor and ignore facet configuration - select this option to have bundles assembled using Bundlor. Bundlor helps you simplify the creation and maintenance of the OSGi metadata of each bundle. In the Bundlor file location text box, specify the path to the file to use. Type the path manually or click the Browse button … and choose the desired location in the dialog that opens .
Maven Synchronization	In this area you can select the following option: Do not synchronize facet settings with Maven - if you select this check box, OGGi facet settings will not be synchronized with Maven. It might be helpful if you want to save facet settings that are different from the settings in pom.xml file when you reimport from the Maven project.

Bundle JAR Tab

Warning! The controls in this tab are not available if you have selected the Create using bnd and ignore facet configuration option in the General tab.

Item	Description
JAR filename	Use this field to specify the name of your bundle JAR file.
JAR output path	Use this area to specify the location of the bundle JAR file. You can select from the following options: – Place in module's output path - select this option to place the OSGi bundle JAR in module's output path. – Place in project-wide OSGi bundle output path - select this option to place the bundle JAR in project's output path that you specified in the OSGi: Project Settings dialog box. – Place in this path - select this option to place the OSGi bundle JAR manually. You can either type the location in text field or click the Browse button … and choose the desired location in the dialog that opens .
Always Rebuild Bundle JAR	Select this check box to have the OSGi bundle JAR file rebuilt upon every module make.
Additional JAR Contents	In this area, specify extra sources to be packaged into the JAR and the target location where to store the packaging results. – Source File/Folder - this read-only field shows the full path to the sources to package. – Destination File/Folder (Relative to JAR Root - this read-only field shows where to place the packaging results. – + (`Alt+Insert`) - use this icon or shortcut to open the Choose Source File or Folder dialog box, where you can specify the desired extra sources to be included in the JAR file. – (`Enter`) - use this icon or shortcut to open the Choose Source File or Folder dialog box, where you can choose a different extra source file or folder. – — (`Alt+Delete`) - use this icon or shortcut to remove the selected source - destination mapping from the list.
File Ignore Pattern (regex)	Use this field to exclude items you do not need from the JAR file after the compilation. Bundles are assembled using bnd tool. This parameter acts as bnd directive `-donotcopy` . For example, you usually do not copy CVS and .svn directories. So, the default, in this case, is `(CVS|.svn)` . The syntax for the parameter is as follows: `-donotcopy= (CVS|.svn|.+.bak|~.+)`

Manifest Generation Tab

Warning! The controls in this tab are only available when the Create manifest using facet settings and bundle using facet configuration option is selected in the General tab.

Item	Description
Symbolic Name	In this text box, specify an alias to refer to the bundle.
Bundle Activator	In this text box, specify the class that implements the OSGi activator. Type the class name or click the Browse button … and choose it in the Select Bundle Activator Class dialog box, that opens.
Bundle Version	In this text box, specify the version of the framework integrator used.
Additional Properties	In this text box, type additional properties to be included in the `manifest.mf` manifest file. Syntax highlighting, auto-completion, and intention actions are available.

Seam Facet Page

File | Project Structure | Modules - module - Seam facet

Ctrl+Shift+Alt+S

Use this page to select the Seam version, and download missing libraries.

Spring Facet Page

File | Project Structure | Modules - module - Spring facet

`Ctrl+Shift+Alt+S`

Use this page to manage Spring configuration file sets .

Icon	Shortcut	Description
+	`Alt+Insert`	Use this icon or shortcut to create a new file set. Select: – XML to create a file set containing XML configuration files. – Annotation Config to create a file set containing `@Configuration`-annotated Java classes. Configure the file set in the Edit Application Context dialog that opens.
	`Enter`	Use this icon or shortcut to edit the selected file set. Modify the file set as required in the Edit Application Context dialog that opens.
–	`Alt+Delete`	Use this icon or shortcut to remove the selected configuration files from a file set, or to delete the selected file set. Note that this operation does not physically delete the corresponding configuration files.

Edit Application Context Dialog

This dialog opens when you click + (Alt+Insert) or (Enter) on the Spring Facet page in the Project Structure dialog.

Use this dialog to specify the properties and the contents of a file set .

Item	Description
Name	Specify the name of the file set.
Parent fileset	If necessary, select a parent file set .
+	Click this button to add a config file and select the file type from the list: – Additional properties files – XML based configurations – Code based configurations Then select the required file in the dialog that opens .
/	Use these buttons to collapse/expand the nodes in the tree view.
Tree view	This area shows the Spring configuration files existing in your module, and the associated libraries. (These may be XML files or `@Configuration` -annotated Java classes.) Use the check boxes to select or deselect the configuration files and groups of files.

Struts Facet Page

File | Project Structure | Modules - module - Struts facet

Ctrl+Shift+Alt+S

Here you can edit settings for the selected Struts facet . The following tabs are available:

– Struts Features tab

– Validation tab

Features tab

Item	Description
Select Struts version	The drop-down list is disabled and shows the Struts version specified during the creation of the dedicated facet.
Struts taglib	The component provides a set of JSP custom tag libraries that help developers create interactive form-based applications. Struts tags help with everything from displaying error messages to dealing with nested ActionForm beans.
Struts El-taglib	The Struts El component expands the Struts taglib component with JSTL (the JavaServer Pages Standard Tag Library) as a foundation technology. This option is available only after you have enabled the Struts taglib component.
Tiles	The component is a templating system that can be used to create a common look and feel for a Web application. Tiles can also be used to create reusable view components.
Validator	The component provides the functionality to validate the form data both on the server side and the client side. The Validator Framework uses two XML configuration files: – `validator-rules.xml` defines the standard reusable validation routines, that are used in `validation.xml` to define the form specific validations. – `validation.xml` defines the validations applied to a form bean.
Struts-Faces	The component supports the use of JavaServer Faces (JSF) user interface technology in a Struts-based Web application.
Scripting	The Struts Scripting component allows Struts Actions to be written in the scripting language of your choice.
Extras	The Struts Extras component provides several popular but non-essential classes.

Validation tab

Item	Description
Disable property keys validation	Excludes Struts property keys from validation.

Struts 2 Facet Page

File | Project Structure | Modules - module - Struts 2 facet

Ctrl+Shift+Alt+S

The page is available only for modules with enabled Struts 2 support.

Use this page to configure the settings for an individual Struts 2 facet attached to a particular module.

The following tabs are available:

- File Sets tab
- Features tab

File Sets tab

Item	Description
(Alt+Insert)	Use this icon or shortcut to create a new file set or add a file to the selected file set. (The Edit File Set dialog will open.)
(Enter)	Use this icon or shortcut to edit the selected file set. (The Edit File Set dialog will open.)
(Alt+Delete)	Use this icon or shortcut to remove the selected file or file set.

Features tab

Item	Description
Struts Version	A drop-down list to select the relevant Struts 2 version.

Tapestry Facet Page

File | Project Structure | Modules - module - Tapestry facet

Ctrl+Shift+Alt+S

For this page to be available, the Tapestry Support plugin must be enabled.

Use this page to specify the configuration settings for your Tapestry application.

Item	Description
Filter Name	Specify the filter name for your application. By convention, the filter name is almost always `app`, but you can use any name you want.
Application Package	Specify the root package for your application.

Web, EJB and Java EE Application Facet Pages

File | Project Structure | Modules - module - Web, EJB or Java EE Application facet

`Ctrl+Shift+Alt+S`

Use this page to edit the settings for a Web, EJB or Java EE Application facet .

Controls for the above mentioned facets are similar, and the differences are specially marked.

Item	Description	Available for
Name	In this text box, type the name of the facet.	Web, JavaEE, EJB facets
Deployment Descriptors	Use the controls in this section to manage application deployment descriptors. – Type - this read-only field shows the type of deployment descriptor, depending on the facet type: Web Module Deployment Descriptor, Application Module Deployment Descriptor, or EJB Module Deployment Descriptor, respectively. – Path - this read-only field shows the location of the deployment descriptor, `web.xml`, `ejb.xml`, or `application.xml`, respectively. – + (`Alt+Insert`) - use this icon or shortcut to create a new deployment descriptor. In the Deployment Descriptor Location dialog box that opens specify the descriptor location and the specification (version) to be used. Only one deployment descriptor can be created per facet. – (`Enter`) - use this icon or shortcut to specify another location for the selected deployment descriptor. – – (`Alt+Delete`) - use this icon or shortcut to remove the selected descriptor from the list. To delete the selected descriptor from disk as well, select the Also delete from disk check box in the Delete Deployment Descriptor confirmation dialog box that opens. – Add Application Server specific descriptor - click this button to add a deployment descriptor for one of the supported application servers (Geronimo, GlassFish, Tomcat, JBoss, or WebLogic). In the Create Application Server Specific Descriptor dialog box that opens specify the desired application server and its version to use.	Web, JavaEE, EJB facets
Web Resource Directories	Use this section to map third party or unclassified resource directories to the deployment directory roots. – Web Resource Directory - this read-only field shows the local directory where the desired Web resource resides. A Web resource directory contains files required for Web development: JSP, HTML, XML, etc. The contents of the Web resource directory are copied to the Web module deployment output according to the Relative Path setting. – Path Relative to Deployment Root - this read-only field shows the path to your Web resource directory relative to the deployment directory root. – + (`Alt+Insert`) - use this icon or shortcut to configure a new mapping in the Web Resource Directories dialog box that opens. – (`Enter`) - use this icon or shortcut to re-configure the selected mapping in the Web Resource Directories dialog box that opens. – – (`Alt+Delete`) - use this icon or shortcut to remove the selected mapping from the list. – ? - click this button to have the reference topic for the Web Resource Directories dialog box displayed.	Web facet
Source Roots	This area shows the list of source roots encountered in the current module. Select the check box next to a root to have the classes from this root included in the corresponding artifact.	Web facet
Create Artifact	This button appears when no artifact of the relevant type is available. Click this button to create a new artifact on the Artifacts page that opens.	Web, JavaEE, EJB facets
Fix	Click this button: – To have the missing libraries downloaded and installed automatically. – To create an artifact of the relevant type and add the facet resources to it on the Artifacts page that opens. – To add the facet resources into an existing artifact on the Artifacts page that opens.	Web, JavaEE, EJB facets

Web Resource Directory Path Dialog

File | Project Structure | Modules - module - Web facet - Web Resource Directories - New

Use this dialog box to add a Web resource directory path to a module with a Web facet and map this path to the deployment directory root.

Item	Description
Web resource directory path	In this text box, specify the local directory where the desired Web resource resides. Type the path manually or click the Browse button and choose the necessary location in the dialog that opens .
Relative path in deployment directory	In this text box, type the directory to deploy the Web resource to relative to the deployment directory root. For example, if you type a forward slash `/`, the files from the Web resource directory will be copied into the deployment root directory.

Web Services Facet Page

File | Project Structure | Modules - module - Web Services facet

Ctrl+Shift+Alt+S

The page is available only for modules with enabled Web Services support.

Use this page to configure the settings of a Web Services facet .

Item	Description
Select WS Engine	Select the archive with the libraries that support the development of a particular Web services type.

Web Services Client Facet Page

File | Project Structure | Modules - module - Web Services Client facet

Ctrl+Shift+Alt+S

The page is available only for modules with enabled Web Services client support.

Use this page to configure the settings of a Web Services Client facet .

Item	Description
Select WS Engine	Select the archive with the libraries that support the development of the client side for Web services of a specific type.

Module Page for a Flash Module

File | Project Structure | Modules

Ctrl+Shift+Alt+S

This page opens in the right-hand part of the Project Structure dialog when you select a Flash module in the element selector pane .

Use this page to edit the module name and configure the module contents.

The module contents are configured by adding and removing the module content roots as well as by assigning individual folders (within the content roots) to source folders, test source folders and also by excluding the folders.

– Name field
– The left-hand pane
– The right-hand pane

Name field

Item	Description
Name	Use this field to edit the module name.

The left-hand pane

The left-hand pane shows a tree of folders for a module content root . If the module has more than one content root, the structure shown corresponds to the content root selected in the right-hand pane.

The folders belonging to different categories have different icons.

The following table lists the available toolbar buttons (the Icon column) and explains their functions (the Description column). Note that the corresponding functions can also be accessed as the context menu commands. These are listed in the Command column.

Most of the icons/commands work as toggles, and can be used to cancel the corresponding assignment (to make a folder "an ordinary folder").

Icon	Command	Description
	Sources	Use this icon or command to assign the selected folder or folders to the source folder category.
	Tests	Use this icon or command to assign the selected folder or folders to test sources.
	Excluded	Use this icon or command to make the selected folder excluded.
	New Folder	Use this command to create a folder in the selected folder. Specify the name for the new folder in the New Folder dialog that opens.

The right-hand pane

The right-hand pane shows the module content roots .

For each content root, a categorized view of the module folders is provided. The categories are the source folders, test source folders, etc.

The "ordinary" folders are not shown in this view.

The individual folders within the categories are identified by their paths. The folder paths are all relative to the module root folder (content root).

The folder paths, functionally, are hyperlinks that let you jump to the corresponding folders in the tree shown in the left-hand pane.

If a module has more than one content root, selecting a content root in the right-hand pane also switches the tree view in the left-hand pane. That is, when you click somewhere within the content root area, the folder structure of this particular content root is shown in the left-hand pane.

The following table lists the controls available in the right-hand pane (icons) and describes their functions.

Icon	Tooltip	Description
	Add Content Root	Use this icon to add a content root. Select the folder to be added as a content root in the dialog that opens .
	Remove Content Entry	Use this icon to remove the corresponding content root from the list of the content roots.
	Unmark	Use this icon to remove the folder from the corresponding category. As a result, the folder becomes "an ordinary folder", that is, not belonging to any specific category.

Build Configuration Page for a Flash Module

File | Project Structure | Modules

Ctrl+Shift+Alt+S

The Build Configuration page opens in the right-hand part of the Project Structure dialog when you select a build configuration of a Flash module in the element selector pane .

Use this page to edit the settings for the selected build configuration.

- General Tab
- Dependencies Tab
- Compiler Options Tab
- AIR Package Tab
- Android Tab
- iOS Tab

General tab

Use this tab to edit such build configuration settings as its name, type , output file name and folder, etc.

Item	Description
Name	Use this field to edit the name of the build configuration.
Type	A brief description of the build configuration type , for example, ActionScript application for web (Flash Player) which means: – The target platform is Web. – The output type is Application (SWF). – The application is pure ActionScript (i.e. it doesn't use the Flex framework). To change the build configuration type, click Change and specify the build configuration properties in the dialog that opens.
Main class	For the Application and the Runtime-loaded module output types: the main class of the SWF file . Edit the name of the class right in the field or click … (Shift+Enter) and select the necessary class in the Choose Main Class dialog.
Output file name	Use this field to edit the name and extension of the output file.
Output folder	The folder in which the output file is generated. Edit the path to this folder right in the field or click … (Shift+Enter) and select the necessary folder in the dialog that opens .
Use HTML wrapper	For Web Applications: select this option if you want an HTML wrapper to be included in the output. – Folder with template. Specify the path to the folder where the files that constitute an HTML wrapper template are located. – To select an existing folder with the wrapper template files, click … (Shift+Enter) and select the folder in the dialog that opens . Note that the corresponding folder must contain the file `index.template.html` and this file must contain the token `${swf}` . – To create a new folder with the wrapper template files, click Create and specify the folder location and the wrapper options in the Create HTML Wrapper Template dialog . During the compilation, the tokens contained in the template (`index.template.html`) are replaced with the appropriate values. For example, `${swf}` is replaced with the `.swf` file name. The resulting `.html` wrapper file will have the same name as the `.swf` file. See also, Using the SWF metadata tag to control HTML wrapper properties .
Runtime-loaded modules	For Web and Desktop Applications: if your application has modular structure (see Modular applications overview in Flex documentation), you can use this field to specify dependencies on the corresponding runtime-loaded modules (RLMs). Click (Shift+Enter). In the Runtime-Loaded Modules Dialog that opens: – To add a dependency, click + (Alt+Insert) and select the main class of the corresponding RLM in the Choose Main Class of Runtime-Loaded Module dialog that opens. – To optimize the module SWF file size, select the Optimize check box. – To replace a class with a different one, click the corresponding Main Class table cell, click … (Shift+Enter), and select the necessary class in the dialog that opens. – To remove dependencies from the list, select the dependencies (RLMs) to be removed and click – (Alt+Delete). Note that there is also an alternative way of specifying dependencies on RLMs: you can add dependencies on RLM build configurations on the Dependencies Tab . See also, Configuring Dependencies for Modular Applications .
Runtime style sheets	For Web and Desktop Applications: if necessary, specify the `.css` files that should be compiled into runtime style sheets . Click (Shift+Enter). In the CSS Files To Compile dialog that opens: – To add a `.css` file to the list, click + (Alt+Insert) and select the necessary file in the dialog that opens . – To replace a file which is already in the list with a different one, click the corresponding entry, click … (Shift+Enter), and select the file in the dialog that opens . – To remove items from the list, select the items to be removed and click – (Alt+Delete).
Skip compilation	Select this option if you do not intend to compile your module using this build configuration. If you do so, the build configuration will only affect your code validation and (error) highlighting.

Dependencies tab

Use this tab to configure the build configuration build path (the build configuration dependencies).

Item	Description
Flex/AIR SDK	The Flex or AIR SDK associated with the build configuration. Select the SDK from the list or click New and select the folder containing the necessary SDK in the dialog that opens . To edit the current SDK, click Edit. (The SDK page of the Project Structure dialog will open.) Note that the necessary SWCs from the specified SDK are selected automatically depending on the build configuration type.
Target player	For the Web target platform only: the version of Flash player the build configuration output is intended for. If the SDK includes more than one player version, you can choose which of the corresponding SWCs should be used.
Component set	For Flex framework-based build configurations (the Web and Desktop output types only): select the Flex 4 component set or sets: – Spark + MX. Select this option to be able to use the Spark and MX components (SWCs) available in the SDK. – Spark only. The names of this and the following option are self-explanatory. – MX only. Note how the list of dependencies changes depending on your selection.
Framework linkage	For Flex framework-based build configurations: select the linkage type for the Flex framework components included in the build configuration dependencies. (The set of available options depends on the build configuration type .)
+ (Alt+Insert)	Use this icon or shortcut to add another dependency. Select: – Build Configuration to add a dependency on a build configuration that generates a library or a runtime-loaded module (RLM). Note that for Web and desktop applications, the dependencies on RLMs can alternatively be specified on the General Tab in the Runtime-loaded modules field . See also, Configuring Dependencies for Modular Applications . – New Library to add a dependency on third-party libraries. Select the libraries of interest in the dialog that opens . – Project or Global Library to add a dependency on a global or project library. Select the libraries of interest in the Choose Libraries dialog.
– (Alt+Delete)	Use this icon or shortcut to remove the selected dependency.
	Click this icon to edit the selected third-party library in the Configure Library dialog.

Compiler Options tab

Use this tab to manage the compiler options for the build configuration as well as the associated project and module defaults.

See the descriptions of the compiler options in Flex documentation:

– For applications

– For libraries

Item	Description
Option	The option name or the name of a group of options. The groups of options are shown as nodes which you can expand or collapse. Note that each option has an associated Restore Default Value context menu command. Use this command to restore the module default for an option. Also note that you can quickly find an option of interest among the options that are currently shown. Click somewhere within the area where the compiler options and their values are shown and start typing the text which, as you expect, is present within the option name. The Search for box appears which contains the text that you are typing. As soon as it's possible to identify an option basing on the text that you have typed, this option is highlighted in the table.
Value	The option value. In most of the cases, to start editing a value, you should click the corresponding table cell. Generally, the way to edit a value depends on which value the corresponding option may have: – The options that you can turn on or off are controlled by check boxes. The actual option values in such cases are `true` or `false`. – If an option value is a string, the value is edited right in the field. – If a value represents a path to a certain location (for example, a path to a file), you can edit such a value right in the field. Alternatively, you can use … (Shift+Enter) to select the necessary location in the corresponding dialog . – If an option value represents a list, such a value cannot be edited directly. To edit the value, use (Shift+Enter). This will open a dedicated dialog for managing the list items.
Legend	The options whose values are inherited from the default sets of different levels (the IDE, project, or module) and the ones having specific values (that is, whose values are redefined at the level of the build configuration) are shown differently. The legend shows how to distinguish between these four categories of options.
Copy resource files to output folder	For Applications: select this check box if you want resource files to be copied to the output folder. Specify which files within the module source roots should be treated as the resource files: – All except *.as and *.mxml. All the files with the extensions other than `.as` and `.mxml` are considered to be the resource files. – According to resource patterns. The files that match the resource patterns are treated as the resource files. Click the resource patterns link to view or edit the resource patterns. (The settings in the dialog that opens correspond to those on the Compiler page in the Settings dialog.) Files and folders excluded from compilation won't be copied to the output folder. See Excludes .
Files to include in output *.swc	For Libraries: if necessary, specify additional files and folders that should be included in the resulting `.swc` file. Click (Shift+Enter). In the Files And Folders To Include In *.swc dialog that opens: – To add a file or folder to the list, click + (Alt+Insert) and select the necessary file or folder in the dialog that opens . – To replace a file or folder which is already in the list with a different file or folder, click the corresponding entry, click … (Shift+Enter), and select the file or folder in the dialog that opens . – To remove items from the list, select the items to be removed and click – (Alt+Delete). Please note the following: – For the source files (`.as` , `.mxml` and `.fxg`) to be included, all such files should be selected individually, as separate files. That is, if you select a folder, none of the `.as` , `.mxml` and `.fxg` files contained therein will be included in the `.swc` file. – Files and folders excluded from compilation won't be included in the `.swc` file. See Excludes .
Additional compiler configuration file	You have an option of specifying a compiler configuration file, an XML file that contains the compiler settings that are used in addition to those specified in the table. The additional compiler configuration file, usually, is a file that you compose yourself. Alternatively, this may be a file generated by means of the Maven plugin, or the like. You can find examples of the configuration files in the directory `<Flex_SDK_directory>\frameworks` . Note that in the case of contradictions, the settings in the configuration file take precedence over the ones specified in the table. To specify the configuration file, click … (Shift+Enter) and select the necessary file in the dialog that opens .
Inherited options	In this field, the compiler options inherited from upper levels (module and project) are shown (readonly).

Additional compiler options	If necessary, specify the compiler options to be used in addition to those in the table and in the additional compiler configuration file.
	When specifying the options, you can use path variables . These include the predefined variables `${MODULE_DIR}` , `${PROJECT_DIR}` and `${USER_HOME}` , and also the ones set on the Path Variables page of the Settings dialog . The format to be used is:
	
	`-some-option=${PATH_VARIABLE_NAME}/relative/path`
	
	For example:
	
	`-dump-config=${MY_PATH_VARIABLE}/config_dump.xml`
	
	To separate individual options, use spaces.
	
	If you need more room to type, click (Shift+Enter) to open the Additional Compiler Options dialog where the text entry area is larger.

Project Defaults	Click this button to edit the project defaults in the Default Compiler Options For Project dialog.
Module Defaults	Click this button to edit the module defaults in the Default Compiler Options For Module dialog.

AIR Package tab

Use this tab to manage the settings related to packaging your Desktop (AIR) application.

Note that this tab is available only if the target platform of the build configuration is Desktop (AIR) and the output type is Application.

Item	Description
Application descriptor	Specify the application descriptor to be used: – Generated. An auto-generated descriptor will be used. – Custom template. The descriptor generated according to the specified template will be used. To use an existing template, click … (Shift+Enter) and select the template file in the dialog that opens . To create and use a new file, click Create. In the Create AIR Descriptor Template dialog that opens, specify the descriptor file properties and click Create. When the template is used for generating the application descriptor (e.g. during the compilation or packaging), the text in the `<content>` element is replaced with the name and extension (`.swf`) of the application file.
Package file name	Specify the name of the resulting package file.
Files and folders to package	In addition to the application SWF file , you may also want other application assets to be packaged. If so, specify the locations of these additional assets. – Path to file or folder. Edit the absolute path to the file or folder where the desired asset resides. Use … (Shift+Enter) to select the file or folder in the corresponding dialog . – Its relative path in package. Specify the relative asset location in the package. – + (Alt+Insert). Use this icon or shortcut to add another asset to the list. Select the asset location in the dialog that opens . Note that for assets within the module source roots, their relative locations in the package, by default, will be set the same as in the source folder. That is, for a file `<src>\images\my_icon.png` , its relative location in the package, by default, will be `images\my_icon.png` . – – (Alt+Delete). Use this icon or shortcut to remove the selected item or items from the list.
Use temporary self-signed certificate	Select this option to sign the package with a pre-installed self-signed certificate. Otherwise, specify the settings related to signing the package (see below).
Keystore file	Specify the path to the keystore file. (This is where a private key and corresponding certificate are stored.) Type the path in the field or click … (Shift+Enter) and select the keystore file in the dialog that opens . (If your keystore type is `PKCS12` , your keystore file, most probably, has the `.p12` extension.)
Keystore type	Specify your keystore type. The default value `PKCS12` corresponds to a keystore file with the `.p12` extension.
More options or Less options	Click the link to show or hide the options described below.
Key alias	Specify the key alias. Note that the alias is not necessary if the keystore contains only one key.
Provider class	Specify the Java Cryptography Architecture (JCA) provider for the specified keystore type.
TSA (time-stamping authority)	Specify the URL of an RFC3161 -compliant time-stamp server to time-stamp the digital signature.

Android tab

Use this tab to manage the settings related to packaging your application for Android.

Note that this tab is available only if the target platform of the build configuration is Mobile (AIR Mobile) and the output type is Application.

Item	Description
Enabled	Turn this option on if you are going to use the build configuration for creating an application descriptor and packaging your application for Android.
Application descriptor	Specify the application descriptor to be used: – Generated. An auto-generated descriptor will be used. – Custom template. The descriptor generated according to the specified template will be used. To use an existing template, click … (Shift+Enter) and select the template file in the dialog that opens . To create and use a new file, click Create. In the Create AIR Descriptor Template dialog that opens, specify the descriptor file properties and click Create. When the template is used for generating the application descriptor (e.g. during the compilation or packaging), the text in the `<content>` element is replaced with the name and extension (`.swf`) of the application file.
Package file name	Specify the name of the resulting package file.
Files and folders to package	In addition to the application SWF file , you may also want other application assets to be packaged. If so, specify the locations of these additional assets. – Path to file or folder. Edit the absolute path to the file or folder where the desired asset resides. Use … (Shift+Enter) to select the file or folder in the corresponding dialog . – Its relative path in package. Specify the relative asset location in the package. – + (Alt+Insert). Use this icon or shortcut to add another asset to the list. Select the asset location in the dialog that opens . Note that for assets within the module source roots, their relative locations in the package, by default, will be set the same as in the source folder. That is, for a file `<src>\images\my_icon.png` , its relative location in the package, by default, will be `images\my_icon.png` . – – (Alt+Delete). Use this icon or shortcut to remove the selected item or items from the list.
Use temporary self-signed certificate	Select this option to sign the package with a pre-installed self-signed certificate. Otherwise, specify the settings related to signing the package (see below).
Keystore file	Specify the path to the keystore file. (This is where a private key and corresponding certificate are stored.) Type the path in the field or click … (Shift+Enter) and select the keystore file in the dialog that opens . (If your keystore type is `PKCS12` , your keystore file, most probably, has the `.p12` extension.)
Keystore type	Specify your keystore type. The default value `PKCS12` corresponds to a keystore file with the `.p12` extension.
More options or Less options	Click the link to show or hide the options described below.
Key alias	Specify the key alias. Note that the alias is not necessary if the keystore contains only one key.
Provider class	Specify the Java Cryptography Architecture (JCA) provider for the specified keystore type.

iOS tab

Use this tab to manage the settings related to packaging your application for iOS.

Note that this tab is available only if the target platform of the build configuration is Mobile (AIR Mobile) and the output type is Application.

Item	Description
Enabled	Turn this option on if you are going to use the build configuration for creating an application descriptor and packaging your application for iOS.
Application descriptor	Specify the application descriptor to be used: – Generated. An auto-generated descriptor will be used. – Custom template. The descriptor generated according to the specified template will be used. To use an existing template, click … (Shift+Enter) and select the template file in the dialog that opens . To create and use a new file, click Create. In the Create AIR Descriptor Template dialog that opens, specify the descriptor file properties and click Create. When the template is used for generating the application descriptor (e.g. during the compilation or packaging), the text in the `<content>` element is replaced with the name and extension (`.swf`) of the application file.
Package file name	Specify the name of the resulting package file.
Files and folders to package	In addition to the application SWF file , you may also want other application assets to be packaged. If so, specify the locations of these additional assets. – Path to file or folder. Edit the absolute path to the file or folder where the desired asset resides. Use … (Shift+Enter) to select the file or folder in the corresponding dialog . – Its relative path in package. Specify the relative asset location in the package. – + (Alt+Insert). Use this icon or shortcut to add another asset to the list. Select the asset location in the dialog that opens . Note that for assets within the module source roots, their relative locations in the package, by default, will be set the same as in the source folder. That is, for a file `<src>\images\my_icon.png` , its relative location in the package, by default, will be `images\my_icon.png` . – – (Alt+Delete). Use this icon or shortcut to remove the selected item or items from the list.
Provisioning profile	Specify the location of your iOS provisioning profile. Type the path in the field or click … (Shift+Enter) and select the provisioning profile in the dialog that opens . For information on provisioning profiles, see How to Create a Provisioning Profile for iPhone , Getting Your Provisioning Certificate to be Recognized by Your iPhone or other online resources.
Keystore file	Specify the path to the keystore file. (This is where a private key and corresponding certificate are stored.) Type the path in the field or click … (Shift+Enter) and select the keystore file in the dialog that opens . (Your keystore file, most probably, has the `.p12` extension.)
Apple iOS SDK	If you want to use a particular Apple iOS SDK to package your application, you can specify the path to that SDK this field. For example, if you have built an extension with the latest iOS SDK, you may want to use that SDK when packaging your application. Type the path in the field or click … (Shift+Enter) and select the SDK installation folder in the dialog that opens .
Additional ADT options	If necessary, specify additional command-line options to be passed to ADT. (ADT is a tool used for application packaging.) For more information, see ADT package command in Adobe AIR documentation. If you need more room to type, click (Shift+Enter) to open the Additional ADT Options dialog where the text entry area is larger.

Libraries and Global Libraries

File | Project Structure | Libraries

File | Project Structure | Global Libraries

Ctrl+Shift+Alt+S

When you select the Libraries or the Global Libraries category in the Project Structure dialog , a list of existing project or global libraries is shown in the element selector pane .

Use the toolbar icons, context menu commands or keyboard shortcuts to manage the libraries (see below).

To view or edit the name and contents of a library, select the library of interest, and use the page to the right of the selector pane.

Toolbar icons, context menu commands and shortcuts

Icon	Command	Shortcut	Description
+	New Project Library or New Global Library	Alt+Insert	Use this icon, command or shortcut to create a new project or global library. See Creating a project or a global library.
−	Delete	Delete	Use this icon, command or shortcut to delete the selected library.
	Copy		Use this icon or command to create a copy of the selected library. See Creating a copy of a project or global library.
	Find Usages	Alt+F7	Use this icon, command or shortcut to find the usages of the selected library in the project.
	Move to Global Libraries		For a project library: use this command to move the selected project library to the global (IDE) level. See Moving a project library to a global level.
	Copy to Project Libraries		For a global library: use this command to create a copy of the selected global library at the project level. See Creating a copy of a global library at the project level.
	Add to Modules		Use this command to add the selected project or global library to dependencies of one or more of your modules . In the dialog that opens, select the corresponding modules.

Project Library and Global Library Pages

File | Project Structure | Libraries

File | Project Structure | Global Libraries

Ctrl+Shift+Alt+S

The Project Library or the Global Library page opens in the right-hand part of the Project Structure dialog when you select a project or global library in the element selector pane .

Use this page to edit the library name and to manage the library contents.

The set of the available controls depends on whether you are working with a Java or ActionScript/Flex library, or a JavaScript library.

- Controls for a Java or ActionScript/Flex library
- Controls for a JavaScript library

Controls for a Java or ActionScript/Flex library

Item	Description
Change Version	This button may be available for a library that implements a certain framework or technology (e.g. JSF, Spring) in cases when IntelliJ IDEA can make version-specific file replacements in the library. In such cases, when you click this button, the Downloading Options Dialog opens in which you can select the necessary library version, and also the files to be downloaded. As a result, the files in the library will be replaced with the downloaded files.
+ (Alt+Insert)	Use this icon or shortcut to add items (classes, sources, documentation, etc.) to the library. In the dialog that opens , select the necessary files and folders. For a Java library, these may be individual `.class` and `.java` files, directories and archives (`.jar` and `.zip`) containing such files as well as directories with Java native libraries (`.dll` , `.so` or `.jnilib`). For an ActionScript/Flex library, these may be raw ActionScript 3 libraries, `.swc` , `.jar` and `.zip` files, the directories containing such files, and so on. IntelliJ IDEA will analyze the selected files and folders, and automatically assign their contents to the appropriate library categories (Classes, Sources, Documentation, Native Library Locations, etc.). When IntelliJ IDEA cannot guess the category (e.g. when you select an empty folder), a dialog will be shown, in which you will be able to specify the category yourself.
	To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.
	Click this icon to make certain library items "excluded" (see Excluded library items). In the dialog that opens, select the items that you want IntelliJ IDEA to ignore (folders, archives and folders within the archives), and click OK.
– (Delete)	When you click this icon or press Delete : – The selected "ordinary" library items are removed from the library. – The selected excluded items (see Excluded library items) become "ordinary" items, i.e. their excluded status is cancelled. The items themselves will stay in the library.

Controls for a JavaScript library

Item	Description
+ (Alt+Insert)	Use this icon or shortcut to add items to the library. Select one of the following options: – Attach Files or Directories. Select this option to add JavaScript files. In the dialog that opens, select the necessary files and folders. These may be individual JavaScript files and the directories containing such files. IntelliJ IDEA will analyze the selected files and folders, and automatically assign the JavaScript files to the appropriate categories. Minified files will be assigned to the Release category; ordinary (uncompressed) files will be assigned to the Debug category. When IntelliJ IDEA cannot guess the category (e.g. when you select an empty folder), a dialog will be shown, in which you will be able to specify the category (Release or Debug) yourself. – Attach Debug Version(s). Select this option to add a single uncompressed JavaScript file or a directory containing such files. Note that IntelliJ IDEA won't analyze the contents of the selected files. If you select a minified JavaScript file or a directory containing minified JavaScript files, the corresponding file or files will still be added to the library. – Attach Release Version(s). Select this option to add a single minified JavaScript file or a directory containing such files. Note that IntelliJ IDEA won't analyze the contents of the selected files. If you select an ordinary (uncompressed) JavaScript file or a directory with such files, the corresponding file or files will still be added to the library. – Specify Documentation URL. Select this option to make external online documentation available in IntelliJ IDEA. Specify the documentation URL in the dialog that opens. – Download Documentation. For jQuery: select this option to download and include jQuery documentation in the library.
	Click this icon to make certain library items "excluded" (see Excluded library items). In the dialog that opens, select the folders that you want IntelliJ IDEA to ignore, and click OK.
– (Delete)	When you click this icon or press Delete : – The selected "ordinary" library items are removed from the library. – The selected excluded items (see Excluded library items) become "ordinary" items, i.e. their excluded status is cancelled. The items themselves will stay in the library.

Facets

File | Project Structure | Facets

Ctrl+Shift+Alt+S

When you select the 'Facets' category in the Project Structure dialog , the existing facets grouped by their types and the Detection option are shown in the element selector pane .

The facet types are at the top hierarchical level. The existing facets are shown within the corresponding type, at the second level.

When you select a facet type, the default settings for this type of facet are shown in the right-hand part of the dialog. These settings are applied when a new facet of the corresponding type is created.

Note that there are no default settings for most of the facet types.

When an individual facet is selected, you can see and configure its settings on the Facet page .

When you select Detection, the Framework Detection page is shown in the right-hand part of the dialog. Use the controls on this page to enable or disable auto-detection for various facet types.

Use the toolbar icons, context menu commands or keyboard shortcuts associated with the facet selector pane to manage the facets (see below).

Toolbar icons, context menu commands and shortcuts

Icon	Command	Shortcut	Description
+	New	Alt+Insert	Use this icon, command or shortcut to create a new facet. Select the facet type, and then the module or modules the new facet should be associated with. Note that certain facets have other facets as their parents. To add such facets, you should create a parent facet first. For more information, see Available Facets and Their Dependencies .
−	Delete	Delete	Use this icon, command or shortcut to delete the selected (individual) facets.
	Expand All	Ctrl+NumPad Plus	Use this icon, command or shortcut to show the facet types and individual facets.
	Collapse All	Ctrl+NumPad −	Use this icon, command or shortcut to show only the facet types. (Individual facets will be hidden.)
	Navigate	F4	Use this command or shortcut to show the selected facet as an element of the corresponding module. (The Modules category will be selected and the facet will be shown in the element selector pane.)

Framework Detection page

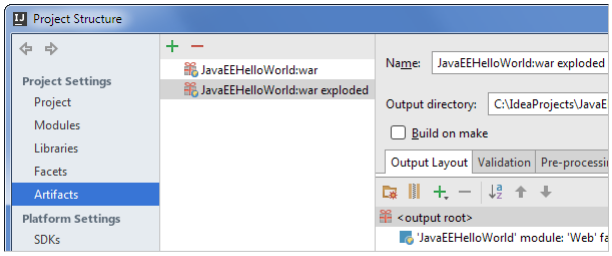
Icon	Shortcut	Description
+	Alt+Insert	Use this icon or shortcut to exclude a facet type from auto-detection . Select the facet type, and then select where auto-detection should be disabled: − In the whole project. − In directory. Select the directory where auto-detection should be disabled in the Select Path dialog that opens. Note that you can disable auto-detection for all the available facet types at once by selecting the All Frameworks option.
−	Alt+Delete	Select the facet types for which you want to restore auto-detection, and click this icon.

Artifacts

File | Project Structure | Artifacts

Ctrl+Shift+Alt+S | Artifacts

| Artifacts



Artifact configurations Artifact configuration settings

- General Settings (Name, Type, etc.)
- Output Layout Tab
- Validation Tab
- Pre-Processing Tab
- Post-Processing Tab
- Android Tab
- Java FX Tab

General settings (Name, Type, etc.)

Item	Description
Name	The name of the artifact configuration and also that of the artifact.
Type	The artifact type. Defines the artifact format and structure, highlighting of problematic parts in the artifact layout as well as the options that IntelliJ IDEA suggests (or assumes acceptable) in relation to the artifact composition.
Output directory	The artifact, when built (Build | Build Artifacts), is placed into the directory whose path is specified in this field. If the artifact is not going to be used on its own but only as a part of other artifacts, the field may be left empty. In this case, the artifact will not appear in the suggestion list on choosing Build | Build Artifacts and no target will be generated for it in `build.xml` on choosing Build | Generate Ant Build.
Build on make	Build the artifact automatically when building the project (Build | Make Project).

Output Layout Tab

File | Project Structure | Artifacts

Ctrl+Shift+Alt+S | Artifacts

| Artifacts

The Output Layout tab lets you specify the artifact structure and contents. The tab includes the following areas:

- The artifact layout pane (the left-hand one) shows the artifact structure and provides the means for changing it. You can create directories ( ) and archives ( ), add copies of compiled module sources, libraries, artifacts, files, etc. ( ) as well as sort ( ) and reorder the items ( ). Note that the order of items, sometimes, may be important .
- The manifest file properties area (in the lower part of the artifact layout pane; available only for JAR, WAR and EAR files) lets you specify which manifest file is associated with the archive. It also lets you view and edit the values for the "most popular" `MANIFEST.MF` header fields.
- The Available Elements pane shows the elements that can be but are not yet added to the artifact. (The top level in this view just provides grouping for the elements.) For adding the elements to the artifact, this pane provides the options that generally work quicker (you can use double-clicks and drag-and-drop) than those in the artifact layout pane.

Reordering the items

The order of items is important when there are elements that lead to producing files with the same name in the same directory. In such situations, only the file from the element that comes first will be included in the artifact.

To see the actual order of items, release . Then use or to reorder the items.

Artifact layout pane: context menu commands

Most of the commands have self-explanatory names. Here, we explain only the commands whose purpose may be unclear.

Command	Description
Extract Artifact	Transform the selected items into a new separate artifact. As a result: 1) a new artifact is created, 2) the selected items are moved to that new artifact, and 3) a copy of the new artifact is included in the old artifact in place of the initially selected items.
Inline Artifact	In a sense, this is the reverse to extracting an artifact and is used to make a "container artifact" independent of the artifact whose copy it contains. Let's assume, artifact A contains a copy of artifact B. As a result of inlining the copy of artifact B: 1) The copy of B in A is replaced with the contents of B and, from this moment on, can be edited independent of B. 2) The artifact B itself remains unchanged.
Surround with	In place of the selected items, create a directory or archive, and move the selected items into that directory or archive.

Validation Tab

File | Project Structure | Artifacts

Ctrl+Shift+Alt+S | Artifacts

| Artifacts

If you are using Oracle WebLogic Server, you can use its deployment descriptor validation extension. Click + and select the server version.

The tab is available only for the artifacts that can be deployed to a server.

Pre-Processing Tab

File | Project Structure | Artifacts

Ctrl+Shift+Alt+S | Artifacts

| Artifacts

If you want IntelliJ IDEA to run an Ant target before building the artifact, select the Run Ant target check box and specify the target. (The corresponding build file must already be added to your project .)

To specify the parameters to be passed to the target, use the Properties section.

Post-Processing Tab

File | Project Structure | Artifacts

Ctrl+Shift+Alt+S | Artifacts

| Artifacts

After the artifact is built, you can process it further by running an Ant target. To do that, select the Run Ant target check box and specify the target. (The corresponding build file must already be added to your project .)

To specify the parameters to be passed to the target, use the Properties section.

Android Tab

File | Project Structure | Artifacts

Ctrl+Shift+Alt+S | Artifacts

| Artifacts

In this tab, configure the Android Application Package to be generated.

Item	Description
Type	Select one of the following options from the drop-down list: – Debug signed with default certificate: select this option to sign the extracted application packaged in the debug mode using the debug keystore or a key that is generated by the Android SDK tools. The following predefined values are used: – Keystore name: `debug.keystore` – Keystore password: `android` – Key alias: `androiddebugkey` – CN (common name): `CN=Android Debug,O=Android,C=US` – Debug signed with custom certificate: Select this option to sign the extracted application packaged in the debug mode using a debug keystore or a key that you specify yourself. You can generate a new certificate, or reuse an existing one. Reusing a certificate may be useful, for example, if you have several applications and you want them all signed with the same certificate to be able to store them in the same folder on an Android device. When this option is selected, the Key store, Key store password, Key alias, and Key password fields become available. – Release unsigned: select this option if you want to run a package on an emulator for test purposes, and want to have it extracted without a release signature. – Release signed: select this option to extract and sign your application so that it can be published an run on physical devices. When this option is selected, the Key store, Key store password, Key alias, and Key password fields become available. Note Note that the debug mode signature is only sufficient for testing and debugging Android applications, and does not allow publishing them.
Key store path	In this text box, specify the location of the file where the key will be stored. Type the path manually or click the Choose existing button to choose the relevant file in the dialog that opens .
Create new	Click this button to open the New Key Store Dialog and configure a new key store and/or a release key to be generated.
Choose existing	Click this button to have your package signed with a key from an existing key store file. Choose the relevant key store file in the dialog that opens . Tip Later you can choose to use an existing key from this key store, or to have a new key generated in it.
Key store password	In this text box, type the password for the selected key store.
Key alias	In this text box, specify the alias to address the key to be used.
Key password	In this text box, specify the password to access the selected key.
Run ProGuard	Select this option if you want IntelliJ IDEA to obfuscate the debug APK through integration with the built-in ProGuard tool.
Config file paths	This text box shows the default location of the `proguard-project.txt` configuration file that is created automatically together with an Android module.
Show content of elements	Select this option if you want the contents of certain elements to be displayed in the layout tree. Click the ... button to specify the elements whose contents you want to be shown: – Show Library Files – Show Content of Included Artifacts – Show Content of JavaEE Facets – Show Content of JPA Resources

Java FX tab

File | Project Structure | Artifacts

Ctrl+Shift+Alt+S | Artifacts

| Artifacts

Use this tab to specify the settings for packaging your JavaFX application or application preloader .

The settings depend on the artifact type (JavaFx Application or JavaFx Preloader).

- JavaFx Application settings
- JavaFx Preloader settings

JavaFx Application settings

Item	Description
Application class	The qualified application main class name. Normally, this is the class that extends the `javafx.application.Application` class and contains the `main()` method.
Title	The application title. In technical terms, this is the information for the `<title>` element in the corresponding deployment descriptor JNLP file . If not specified, the title `Sample JavaFX Application` is used.
Vendor	The name of the application vendor (i.e. the information for the `<vendor>` element in the JNLP file). If not specified, the text `Unknown vendor` is used.
Description	A brief description of the application (i.e. the information for the <description> element in the JNLP file). If not specified, the text `Sample JavaFX 2.0 application` is used.
Width	The width of the application window in pixels. This parameter is used mainly by the applications embedded in a Web page.
Height	The height of the application window in pixels. This parameter is used mainly by the applications embedded in a Web page.
HTML Parameters	You may want to pass dynamic parameters to the application that runs in the Web Start or embedded in browser modes from the corresponding HTML page . In that case, you should create a `.properties` file and specify the necessary set of named parameters in that file. Then you should specify the path to the `.properties` file in this field.

Application Parameters	You may want to pass named and unnamed parameters to the application. In that case, you should create a `.properties` file and specify the necessary parameters in that file. Then you should specify the path to the `.properties` file in this field. (The specified parameters will be included in the generated deployment descriptor JNLP file .)
	Example
	If the `.properties` file contains
	`arg1=value1`
	`arg2`
	the following elements will be generated within `<jfx:javafx-desc>`:
	`<fx:param name="arg1" value="value1"/>`
	`<fx:argument>arg2</fx:argument>`

Update in background	Use this check box to set the `check` attribute of the `<update>` element in the JNLP file. (This element is used to specify how Java Web Start should handle the application updates.) If selected, `<update check="background"/>`. The JNLP client will check for updates in the background while the application is being launched. If not selected, `<update check="always"/>`. The JNLP client will always check for updates before launching the application.

Native bundle	The native bundles to be generated. (A native bundle is an operating system-specific self-contained application package. Such a bundle contains an application itself as well as a JRE, JavaFX runtime and a platform-specific application launcher.)
	Select:
	– none. No OS-specific bundles are created. The application is packaged into the following files:
	  – `<artifact_name>.jar.` This file contains the compiled class files and images.
	  – `<artifact_name>.jnlp.` This is a deployment descriptor JNLP file for Web deployment modes (Web Start and embedded in browser).
	  – `<artifact_name>.html.` This file contains basic code for running the application in the Web Start and embedded in browser modes.
	– all. All the bundles listed below are generated.
	– deb. A Debian software package for Debian GNU/Linux is generated.
	– dmg. An Apple disk image file for OS X is generated.
	– exe. An executable file for Windows is generated.
	– image. A bundle image for the OS that you are using is generated.
	– msi. A Windows Installer (Microsoft Installer) file is generated.
	– rpm. A Red Hat Package Manager file for Linux is generated.

Convert css to bin	Select this check box if you want your application CSS file to be converted into binary format. (This may improve the application performance, especially for "large" CSS files.)

Enable signing	Select this check box if you want the application package to be digitally signed.
	IntelliJ IDEA can generate a key and the corresponding self-signed certificate, or an existing key may be used to sign the package.
	To select which way of signing should be used, click Edit Certificates. Then, in the Choose Certificate dialog that opens, select:
	– Self signed. The package will be signed with the generated self-signed certificate.
	– Signed by key. The package will be signed with your private key. Specify the associated settings:
	  – Alias. Specify your private key alias.
	  – Keystore. Specify the path to the keystore file. (This is where your private key and corresponding certificate are stored.)
	  – Storepass. Specify the password for accessing the keystore.

- Keypass. Specify the password for accessing the key.

JavaFx Preloader settings

Item	Description
Preloader class	The qualified application preloader class name.

SDKs

File | Project Structure | SDKs

Ctrl+Shift+Alt+S

When you select SDKs in the left-hand pane of the Project Structure dialog, the SDKs defined in IntelliJ IDEA are shown. So you can add and remove SDKs, look for their usages in the project as well as manage their settings.

- Adding and removing SDKs. Looking for SDK usages
- Managing SDK settings

Adding and removing SDKs. Looking for SDK usages

The corresponding functions are accessed by means of the toolbar icons (shown in the Icon column). Alternatively, context menu commands (accessed by right-clicking an SDK in the list; listed in the Command column) or keyboard shortcuts (the Shortcut column) may be used.

Icon	Command	Shortcut	Description
+	Add new SDK	Alt+Insert	Use this icon, command or shortcut to add a new SDK. Select the necessary SDK type . Then, in the dialog that opens , select the SDK home directory and click OK.
−	Delete	Delete	Use this icon, command or shortcut to remove the selected SDKs from the list.
	Find Usages	Alt+F7	Use this icon, command or shortcut to look for the usages of the selected SDK in the project.

Managing SDK settings

The settings for the selected SDK are shown in the right-hand part of the dialog on the SDK page. These settings depend on the SDK type .

- SDKs. Flex
- SDKs. Flexmojos SDK
- SDKs. Java
- SDKs. IntelliJ IDEA
- SDKs. Mobile

SDKs. Flex

File | Project Structure | SDKs | Flex SDK

Ctrl+Shift+Alt+S

Use this page to configure the selected Flex SDK . This includes managing the corresponding path lists on the Classpath, Sourcepath and Documentation Paths tabs.

Item	Description
Name	Use this field to edit the name of the SDK.
Flex SDK home path	Specify the Flex SDK home directory. Use ... (Shift+Enter) to select the SDK installation directory in the Select Home Directory for Flex SDK dialog.
+ (Alt+Insert)	Use this icon or shortcut to add one or more files and/or folders to the list. In the dialog that opens , select the necessary files and/or folders and click OK.
– (Alt+Delete)	Use this icon or shortcut to remove the selected items from the list.
(Alt+S)	For documentation: To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.

SDKs. Flexmojos SDK

File | Project Structure | SDKs | Flexmojos SDK

Ctrl+Shift+Alt+S

When importing a Flexmojos project, IntelliJ IDEA automatically creates a Flexmojos SDK that lets you launch the Flex compiler and debugger.

Use the Flexmojos SDK page to view and edit the Flexmojos SDK settings.

Item	Description
Name	Use this field to edit the name of the SDK.
Flex Compiler POM	In this field, the path to your compiler `pom` file is shown (readonly).
Flex Compiler/Debugger Classpath	In this area, the files included in the compiler shell classpath are shown (readonly).
AIR Debug Launcher	Specify the path to the `adl.exe` file.
AIR Runtime (directory or zip file)	Specify the path to the AIR runtime directory or archive.

SDKs. Java

File | Project Structure | SDKs | Java SDK

Ctrl+Shift+Alt+S

Use this page to configure the selected Java SDK . This includes specifying the paths to the class files, sources , external annotations and documentation.

Item	Description
Name	Use this field to edit the name of the SDK.
JDK home path	Specify the JDK home directory. Use ... (Shift+Enter) to select the SDK installation directory in the Select Home Directory for JDK dialog.
+ (Alt+Insert)	Use this icon or shortcut to add one or more files and/or folders to the list. In the dialog that opens , select the necessary files and/or folders and click OK.
– (Alt+Delete)	Use this icon or shortcut to remove the selected items from the list.
(Alt+S)	For documentation: To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.

SDKs. IntelliJ IDEA

File | Project Structure | SDKs | IDEA SDK

Ctrl+Shift+Alt+S

Use this page to specify the settings related to the IntelliJ Platform Plugin SDK . (This is an SDK type for IntelliJ IDEA plugin development .)

Configure the paths to the class files, sources , external annotations and documentation.

Item	Description
Name	Use this field to edit the name of the SDK.
IntelliJ Platform Plugin SDK home path	Specify the IntelliJ Platform Plugin SDK home directory. Actually, this is the IntelliJ IDEA installation directory. Use ... (Shift+Enter) to select the SDK installation directory in the Select Home Directory for IntelliJ Platform Plugin SDK dialog.
Sandbox Home	Specify the sandbox directory, where IntelliJ IDEA will deploy plugins. Use ... (Shift+Enter) to select the necessary directory in the Sandbox Home dialog.
Internal Java Platform	Select the internal Java SDK.
+ (Alt+Insert)	Use this icon or shortcut to add one or more files and/or folders to the list. In the dialog that opens , select the necessary files and/or folders and click OK.
– (Alt+Delete)	Use this icon or shortcut to remove the selected items from the list.
(Alt+S)	For documentation: To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.

SDKs. Mobile

File | Project Structure | SDKs | Java ME SDK

Ctrl+Shift+Alt+S

Use this page to configure the selected Java ME SDK (Mobile SDK). This includes specifying the paths to the class files, sources , external annotations and documentation.

Note that Java ME support is available in the Community edition of IntelliJ IDEA via a free plugin.

Item	Description
Name	Use this field to edit the name of the SDK.
Mobile SDK home path	Specify the Mobile SDK home directory. Use ... (Shift+Enter) to select the SDK installation directory in the Select Home Directory for Mobile SDK dialog.
Use default profile and configuration versions	Choose this option to have your application developed in compliance with the default set of Mobile Information Device Profiles (MIDP) and Connected Limited Device Configurations (CLDC) . This ensures that your application will fit maximum number of targeted platforms. The supported MIDP and CLDC are listed in a read-only field.
Use custom profile and configuration versions	Choose this option to have your application developed in compliance with Mobile Information Device Profiles (MIDP) and Connected Limited Device Configurations (CLDC) of your choice. In this case, your application will fit a restricted number of targeted platforms. Specify the required MIDP and CLDC in the text boxes.
Preverify parameters	Specify the parameters to be passed to the preverify utility . This utility checks your compiled code for compliance with the targeted environment.
Choose Java SDK	Select the internal Java SDK, if any is registered with IntelliJ IDEA.
+ (Alt+Insert)	Use this icon or shortcut to add one or more files and/or folders to the list. In the dialog that opens , select the necessary files and/or folders and click OK.
– (Alt+Delete)	Use this icon or shortcut to remove the selected items from the list.
(Alt+S)	For documentation: To be able to use external documentation available online, click this icon and specify the URL of the external documentation in the dialog that opens.

Keyboard Shortcuts and Mouse Reference

Using shortcuts is a major way to maximum efficiency and productivity of IntelliJ IDEA. This part lists keystroke combinations and their functions for the Default keymap defined in Keymap dialog box.

Warning! The key combinations documented in this part may fail to perform the function described, if you are using a customized keymap .

To get a printable copy of the default keymap, choose Help | Default Keymap Reference on the main menu.

Tip Note that in certain operating systems the key and mouse combinations may not work as described here. In this case, it's necessary to tweak the operating system's keymap. For example, if you are using Ubuntu, mind the windows manager, whose shortcuts conflict with that of IntelliJ IDEA.

In this part:

- Keyboard Shortcuts by Keystroke
- Keyboard Shortcuts by Category
- Mouse Reference

Keyboard Shortcuts By Keystroke

In this part you can find reference information on keyboard shortcuts grouped by keystroke:

- Alt
- Alt+Shift
- Ctrl
- Ctrl+Alt
- Ctrl+Shift
- Function Keys
- Insert, Delete and Navigation Keys
- Shift
- Ctrl+Alt+Shift

To view a full list of available shortcuts, navigate to File | Settings and click Keymap under IDE Settings.

Alt

This section lists and describes the keyboard shortcuts that include the `Alt` key:

- Alt+Alphanumeric keys
- Alt+Navigation keys
- Alt+Function (F) keys

Alt+Alphanumeric keys

Shortcut	Function	Use this shortcut to...
`Alt+O`	Export to Text File	Export a tool window's content to a text file.
`Alt+Q`	Context Info	Show the current method or class declaration when it is not visible.
`Alt+Number`	Open tool window	Open a tool window with the corresponding number.
`Alt+Slash`	Code completion / Expand word	Expand string at caret to any word in the visible scope that starts with the same characters.
`Alt+Back Quote`	VCS operations	Show quick list with the most required version control commands.

Alt+Navigation keys

Shortcut	Function	Use this shortcut to...
`Alt+Delete`	Safe Delete	Delete selected class/method/field checking its usages.
`Alt+Enter`	Show Intention Action	Display intention actions (if any) for a code where the caret is currently located.
`Alt+Home`	Activate Navigation Bar	Bring focus to the Navigation bar .
`Alt+Insert`	Create new entity	Depending on the context: – In the navigation views: create a new class, file or directory , using the New pop-up menu. – In the editor: create constructors, accessor methods, EJB components, Maven dependencies, and test methods, using the Generate pop-up menu. See Generating Code , Creating Maven Dependencies , Creating Test Methods .
`Alt+Down`	Navigate to Next Method	Navigate to the next method declaration in the active editor tab.
`Alt+Left`	Select Previous Tab	Depending on the context: – When several tabs are opened in the editor or a view, open the next tab to the left (or the last tab if the current one is the first). – In the Differences Viewer for Files invoked from the Update Project Info tab of the Version Control tool window, compare the local copy of the previous file with its update from the server.
`Alt+Right`	Select Next Tab	Depending on the context: – When several tabs are opened in the editor or a view, open the next tab to the right (or the first tab if the current one is the last). – In the Differences Viewer for Files invoked from the Update Project Info tab of the Version Control tool window, compare the local copy of the next file with its update from the server.
`Alt+Up`	Navigate to Previous Method	Navigate to the previous method declaration in the active editor tab.

Alt+Function (F) keys

Shortcut	Function	Use this shortcut to...
`Alt+F1`	Select Target	Move focus from the current file, class, method or reference to a data source table to a view suggested in the Select Target pop-up menu. See Navigating Between IDE Components .
`Alt+F7`	Find Usages	Initiate search for usages .
`Alt+F8`	Evaluate Expression	Debugger: Evaluate an arbitrary expression .
`Alt+F9`	Run to Cursor	Debugger: Run to the line where the caret is located.

Alt+Shift

This section lists and describes the keyboard shortcuts that include the `Shift+Alt` keys:

Shortcut	Function	Use this shortcut to...
`Shift+Alt+F7`	Force Step Into	Step into the method called in the current execution point, even if this method is to be skipped.
`Shift+Alt+F8`	Force Step Over	Run until the next line in this method or file, skipping the methods referenced at the current execution point and ignoring breakpoints.
`Shift+Alt+F9`	Debug	Quickly select run/debug configuration and debug/edit it.
`Shift+Alt+F10`	Run	Quickly select run/debug configuration and run/edit it.

Ctrl

This section lists and describes the keyboard shortcuts that include the `Ctrl` key:

- Ctrl+Alphanumeric keys
- Ctrl+Navigation keys
- Ctrl+Symbol keys
- Ctrl+Numpad keys
- Ctrl+Function (F) keys

Ctrl+Alphanumeric keys

Shortcut	Function	Use this shortcut to...
`Ctrl+A`	Select All	Select the entire text in the active editor.
`Ctrl+B`	Navigate to Declaration	Navigate directly to an element's declaration from any usage.
`Ctrl+C`	Copy	Copy selected text to the Clipboard.
`Ctrl+D`	Duplicate Line or Block	Duplicate selected block or line at caret.
`Ctrl+E`	Recent Files Recent find usages	Show the list of recently opened files . When the Find tool window has the focus, use this shortcut to show the list of recent find usages results .
`Ctrl+F`	Find	Initiate text search in the editor.
`Ctrl+G`	Navigate to Line	Navigate the to a line with the specified number in the current file.
`Ctrl+H`	Type Hierarchy	Browse hierarchy for the selected class.
`Ctrl+I`	Implement Methods	Implement methods of the base interface/class in the current class.
`Ctrl+J`	Insert Live Template	Show a pop-up list of Live Templates starting with a specified prefix.
`Ctrl+M`	Scroll to Center	Scroll a line at caret to the center of the screen.
`Ctrl+N`	Navigate to Class	Jump to a class in the project with the specified name.
`Ctrl+O`	Override Methods	Override base class methods in the current class.
`Ctrl+P`	Parameter Info	Show parameters of the method call at the caret.

`Ctrl+Q`
 	
`Alt+Mouse Button2`
	Quick documentation	Show a pop-up window with documentation for the symbol at caret.
		In the Database tool window : show a pop-up window that displays the `create table` query for the database table at the caret and the first 10 rows of the table.

`Ctrl+R`	Replace	Call the Replace Text dialog box.
`Ctrl+S`	Save All	Save all files and settings.
`Ctrl+U`	Navigate to Super Method	Navigate to a super method declaration of a method at caret
`Ctrl+W`	Select Word at Caret	Successively select expanding blocks of text, starting from the word at caret. (Use this shortcut repeatedly to select expressions.)
`Ctrl+Y`	Delete Line at Caret	Delete a word starting from the current caret location up to the end of word.
`Ctrl+Z`	Undo	Undo last operation.
`Ctrl+Shift+Z`	Redo	Redo last undone operation.
`Ctrl+Number`	Navigate to bookmark	Navigate to a numbered bookmark with corresponding number.

Ctrl+Navigation keys

Shortcut	Function	Use this shortcut to...
`Ctrl+Tab`	Switcher	Navigate between the files opened in the editor, and tool windows.
`Ctrl+Backspace`	Delete to Word Start	Delete a word starting from the current caret location up to the word start.
`Ctrl+Delete`	Delete to Word End	Delete a word starting from the current caret location up to the word end.
`Ctrl+End`	Move to Text End	Move the caret to the end of text.
`Ctrl+Enter`	Split Line or Open Item	Depending on the context: – In the editor: Intelligently split the current line into 2 lines, shifting quotes, etc. as necessary. – In the Tool Windows: Open an Editor tab or tabs for the selected item or items, respectively. – On the context menus of the modules in the Project tool window , Dependency Viewer , and Module Dependencies tool window : open the Modules structure .
`Ctrl+Home`	Move to Text	Jump to the beginning of the text.

	Start	
Ctrl+C	Copy	Copy a current line or a selected code block to the Clipboard.
Ctrl+Space	Basic Code Completion	Complete code for any class, method or variable.
Ctrl+Page Down	Navigate to Page Bottom	Move the caret down to the page bottom.
Ctrl+Page Up	Navigate to Page Top	Move the caret up to the page top.
Ctrl+Down	Scroll Down	Move line at caret one down, preserving syntactical correctness.
Ctrl+Left	Move to Previous Word	Move the caret to the previous word.
Ctrl+Right	Move to Next Word	Move the caret to the next word.
Ctrl+Up	Scroll Up	Move line at caret one up, preserving syntactical correctness.
Ctrl+Shift+Up		
Ctrl + Ctrl + Home / End		Select text from the caret position to the beginning/end of the current line.

Ctrl+Symbol keys

Shortcut	Function	Use this shortcut to...
Ctrl+Open Bracket	Move to Code Block Start	Move the caret to the beginning of the current code block, highlighting its limits.
Ctrl+Close Bracket	Move to Code Block End	Move the caret to the end of the current code block, highlighting its limits.
Ctrl+Slash	Comment with Line Comment	Comment/uncomment current line or selected block with line comments.
Ctrl+Numpad/		
Ctrl+=	Expand All	Expand all folding blocks.
Ctrl+NumPad Plus		
Ctrl+NumPad -	Collapse All	Collapse all folding blocks.

Ctrl+Numpad keys

Shortcut	Function	Use this shortcut to...
Ctrl+Numpad/	Comment with Line Comment	Comment/uncomment current line or selected block with line comments.
Ctrl+Slash		
Ctrl+NumPad Plus	Expand All	Expand all folding blocks.
Ctrl+=		
Ctrl+NumPad -	Collapse All	Collapse all folding blocks.

Ctrl+Function (F) keys

Shortcut	Function	Use this shortcut to...
Ctrl+F1	Error Description	Show an error or warning description at the caret.
Ctrl+F3	Find Word at Caret	Search in the editor for the word where the caret is currently located.
Ctrl+F6	Change Method Signature	Refactor a selected method signature and update all references.
Ctrl+F7	Find Usages in File	Initiate search for usages .
Ctrl+F8	Toggle Breakpoint	Toggle breakpoint at caret.
Ctrl+F9	Make Project	Compile all modified and dependent files in a project.
Ctrl+F11	Toggle Bookmark with mnemonic.	Turn bookmark with mnemonic on or off.
Ctrl+F12	File Structure Pop-up	Show the current file structure in the File Structure pop-up window for quick navigation.

Ctrl+Alt

This section lists and describes the keyboard shortcuts that include the `Ctrl+Alt` keys:

– Ctrl+Alt+Alphanumeric keys
– Ctrl+Alt+Navigation keys
– Ctrl+Alt+Function (F) keys

Ctrl+Alt+Alphanumeric keys

Shortcut	Function	Use this shortcut to...
`Ctrl+Alt+B`	Navigate to Implementation	Navigate to implementation of an item at the caret.
`Ctrl+Alt+C`	Extract Constant	Replace selected expression with a constant (static final field) (Refactoring).
`Ctrl+Alt+F`	Extract Field	Put a selected expression result into a field (Refactoring).
`Ctrl+Alt+G`	Run Grails target	Execute Grails target with the specified target name.
`Ctrl+Alt+H`	Call Hierarchy	Browse call hierarchy for the selected method.
`Ctrl+Alt+I`	Auto-indent Lines	Indent current line or selected block according to the Code Style Settings.
`Ctrl+Alt+J`	Surround with Live Template	Surround the selection with one of the Live Templates.
`Ctrl+Alt+M`	Extract Method	Create a method from the selected code (Refactoring).
`Ctrl+Alt+N`	Inline	Inline a selected method/variable (Refactoring).
`Ctrl+Alt+P`	Extract Parameter	Turn a selected expression into a method parameter (Refactoring).
`Ctrl+Alt+T`	Surround with	Surround selected code fragment with if, while, try/catch, or another construct.
`Ctrl+Alt+V`	Extract Variable	Put selected expression result into a variable (Refactoring).
`Ctrl+Alt+Y`	Synchronize	Detect all externally changed files and reload them from disk.

Ctrl+Alt+Navigation keys

Shortcut	Function	Use this shortcut to...
`Ctrl+Alt+Enter`	Start new line before current one	Start a new line before the current one.
`Ctrl+Alt+Down` `Ctrl+Alt+Up`	Navigate to Next/Previous Occurrence	Navigate to a next found item .
`Ctrl+Alt+Left`	Back	Undo last navigation operation. Note On an OS X computer, you can also use the three-finger right-to-left swipe gesture.
`Ctrl+Alt+Right`	Forward	Redo last undone navigation operation. Note On an OS X computer, you can also use the three-finger left-to-right swipe gesture.
`Ctrl+Alt+Home`	Navigate to Related Symbol	Navigates between files with the various relationships. See Navigation in Source Code .

Ctrl+Alt+Function (F) keys

Shortcut	Function	Use this shortcut to...
`Ctrl+Alt+F6`	Switch to another coverage suite.	Open the Coverage Suites popu-up window and select the desired suite to run.
`Ctrl+Alt+F7`	Show usages	Show usages of a symbol at the caret.
`Ctrl+Alt+F8`	Quick Evaluate Expression	Evaluate an arbitrary expression without calling Evaluate Expression dialog box.
`Ctrl+Alt+F9`	Force Run To Cursor	Run to the line where the caret is located, ignoring breakpoints.

Ctrl+Shift

This section lists and describes the keyboard shortcuts that include the `Ctrl+Shift` keys:

- Ctrl+Shift+Alphanumeric keys
- Ctrl+Shift+Navigation keys
- Ctrl+Shift+Symbol keys
- Ctrl+Shift+Numpad keys
- Ctrl+Shift+Function (F) keys

Ctrl+Shift+Alphanumeric keys

Shortcut	Function	Use this shortcut to...
`Ctrl+Shift+A`	Find Action	Find an action, bypassing menus. See Finding Actions .
`Ctrl+Shift+B`	Navigate to Type Declaration	Navigate to type declaration of a variable or a method call at caret.
`Ctrl+Shift+E`	Navigate to Recently Changed File	Show the list of recently updated files .
`Ctrl+Shift+F`	Find in Path	Initiate text search in the specified path .
`Ctrl+Shift+H`	Method Hierarchy	Browse hierarchy for the selected class.
`Ctrl+Shift+J`	Join Lines	Concatenate selected lines into one or concatenate a line where the caret is currently located with the next line.
`Ctrl+Shift+N`	Navigate to File	Jump to the specified file in project.
`Ctrl+Shift+R`	Replace in Path	Initiate text replacement in the specified path .
`Ctrl+Shift+T`	Navigate to Test/Test Subject	Navigate to a test for the class at caret, if any, or create a new test class. For a test class, navigate to its test subject.
`Ctrl+Shift+U`	Toggle Case	Toggle case of the selected text fragment.
`Ctrl+Shift+V`	Paste from History	Paste from recent Clipboards.
`Ctrl+Shift+W`	Deselect Word at Caret	Remove sequential selection made by the Select Word at Caret action .
`Ctrl+Shift+Z`	Redo	Redo the last Undo operation .

Ctrl+Shift+Navigation keys

Shortcut	Function	Use this shortcut to...
`Ctrl+Shift+End`	Move to Text End with Selection	Select text from the current caret position to the end of text, and move caret to the end of text.
`Ctrl+Shift+Home`	Move to Text Start with Selection	Select text from the current caret position to the start of text, and move caret to the start of text.
`Ctrl+Shift+Right`	Move to Word End with Selection	Select text from the current caret position to the end of word, and move caret to the end of word.
`Ctrl+Shift+Left`	Move to Word Start with Selection	Select text from the current caret position to the beginning of the current word, and move caret to to the beginning of this word.
`Ctrl+Shift+V`	Paste from History	Paste from recent Clipboards.
`Ctrl+Shift+Space`	SmartType Code Completion	Complete code, filtering the lookup list based on an expected type.
`Ctrl+Shift+Page Down`	Navigate to Page Bottom with Selection	Move the caret down to the page bottom selecting the text.
`Ctrl+Shift+Page Up`	Navigate to Page Top with Selection	Move the caret up to the page bottom selecting the text
`Ctrl+Shift+Down`	Move Line Down	Move line at caret one down, preserving syntactical correctness.
`Ctrl+Shift+Up`	Move Line Up	Move line at caret up, preserving syntactical correctness.
`Ctrl+Shift+Backspace`	Last Edit Location	Jump to the place of the last editing.

Ctrl+Shift+Symbol keys

Shortcut	Function	Use this shortcut to...
`Ctrl+Shift+Open Bracket`	Move to Code Block Start with Selection	Move the caret to the beginning of the current code block, selecting the code from the initial caret location.
`Ctrl+Shift+Close Bracket`	Move to Code Block End with Selection	Move the caret to the end of the current code block, selecting the code from the initial caret location.
`Ctrl+Shift+Slash`	Comment with Block Comment	Comment/uncomment code with block comments.
`Ctrl+NumPad Plus`	Expand All	Expand all folding blocks.
`Ctrl+Shift+NumPad -`	Collapse All	Collapse all folding blocks.

Ctrl+Shift+Numpad keys

Shortcut	Function	Use this shortcut to...
`Ctrl+Shift+Numpad/`	Comment with Block Comment	Comment/uncomment code with block comments.
`Ctrl+Shift+Numpad+`	Expand All	Expand all folding blocks.

Ctrl+NumPad Plus		
Ctrl+Shift+Numpad-	Collapse All	Collapse all folding blocks.

Ctrl+Shift+Function (F) keys

Shortcut	Function	Use this shortcut to...
Ctrl+Shift+F4	Close Active Tab	Close an active tab in a tool window
Ctrl+Shift+F7	Highlight Usages in File / Highlight Method Exit Points	Highlight usages of a symbol where the caret is currently located. If the caret is placed on one of the method's exit points, like `return`, all method exit points are highlighted.
Ctrl+Shift+F8	View Breakpoints	View/manage all breakpoints/watchpoints
Ctrl+Shift+F9	Compile	Compile the selected file or package.

Function Keys

This section describes default mappings for the function (F) keys.

Shortcut	Function	Use this shortcut to...
F1	Help	Invoke reference page.
F2	Activate in-place editing	In a GUI Designer form, enable in-place editing of the name of a selected UI component.
F3 Shift+F3	Search for next/previous occurrence	Navigate to the next/previous occurrence of a selected word in the editor.
F4	Edit Source	Depending on the context: – In Tool Windows: Open an Editor tab or tabs for the selected item or items (including GUI forms), and give focus to the last opened file. – On the context menus of the modules in the Project tool window , Dependency Viewer , and Module Dependencies tool window : open the Modules structure .
F5	Copy	Create a copy of a selected class/file/directory in the same or a different package.
F6	Move	Move a selected class/package/static member to another package/class and correct all references.
F7	Step Into	Step to the next executed line (during debugging).
F8	Step Over	Step to the next line in the current file (during debugging).
F9	Resume Program	Resume program execution (during debugging).
F11	Toggle Bookmark	Turn anonymous bookmark on or off.
F12	Jump to Last Window	Activate a last focused tool window.

Insert, Delete and Navigation Keys

This section describes default mappings for the `Insert`, `Delete` and the navigation keys.

Shortcut	Function	Use this shortcut to...
`Delete`	Delete	Depending on the context: – In the editor: delete selected symbol/block. – In the Find tool window: exclude items from the search results. – In the Version Control tool window: delete an item from a changelist. – In other views: remove the selected item or items.
`Down`	Move down	Move the caret one line down.
`End`	Move to Line End	Move the caret to the end of line.
`Home`	Move to Line Start	Move the caret to the beginning of line.
`Insert`	Toggle Insert/Overwrite	Toggle Insert/Overwrite modes in the editor. The shape of the cursor changes according to the current mode.
`Left`	Move left	Move the caret one character to the left.
`Page Down`	Page down	Move the caret one page up.
`Page Up`	Page up	Move the caret one page up.
`Right`	Move right	Move the caret one character to the right.
`Tab`		In the editor: – With any selection: indent selected line(s). – Without any selection: insert a tab symbol (or corresponding number of space.characters). In a lookup list: – No code after the caret in the editor: select an item (like `Enter`) – Some code after the caret in the editor: select an item and replace the code after the caret with it.
`Up`	Move up	Move the caret one line up.

Shift

This section lists and describes the keyboard shortcuts that include the `Shift` key:

- Shift+Navigation keys
- Shift+Function (F) keys

Shift+Navigation keys

Shortcut	Function	Use this shortcut to...
`Shift+Down`	Down with Selection	Move the caret one line down selecting the text.
`Shift+End`	Move to Line End with Selection	Move the caret to the end of line, selecting text.
`Shift+Enter`	Start New Line	Start a new line after the current one, positioning the caret in accordance with the current indentation level (equal to sequential pressing End, Enter).
`Shift+Escape`	Hide Active Window	Hide the currently active tool window.
`Shift+Home`	Move to Line Start with Selection	Move the caret to the beginning of line, selecting the text.
`Shift+Left`	Left with Selection	Move the caret one character to the left selecting the text.
`Shift+Page Down`	Page Down with Selection	Move the caret one page down selecting the text.
`Shift+Page Up`	Page Up with Selection	Move the caret one page up selecting the text.
`Shift+Right`	Right with Selection	Move the caret one character to the right selecting the text.
`Shift+Tab`	Unindent Selection	Move selected block to the previous indent level.
`Shift+Up`	Up with Selection	Move the caret one line up selecting the text.

Shift+Function (F) keys

Shortcut	Function	Use this shortcut to...
`Shift+F1`	External Documentation	Open browser with the documentation for the selected item. Refer to Viewing Inline Documentation for details.
`Shift+F2`	One of the following: – Navigate to Previous Highlighted Error. – Stop Program.	Depending on whether you are editing or debugging: – When editing: Navigate to the previous found error/warning. – When debugging: Terminate the debugging session.
`F3` / `Shift+F3`	Search for next/previous occurrence	Jump to the next/previous occurrence of the selected word in the editor.
`Shift+F6`	Rename	Rename a statement and correct all references. (Refactoring).
`Shift+F7`	Move to Previous Difference/Smart Step Into	Move to a previous difference in a view./ Select the method to step in, if the current line contains multiple method call expressions. (Debugger).
`Shift+F8`	Step Out	Step to the first executed line after returning from a current method.
`Shift+F9`	Debug	Debug application.
`Shift+F10`	Run	Run application.
`Shift+F11`	Show Bookmarks	Open Bookmarks dialog to manage existing bookmarks and navigate between them.
`Shift+F12`	Restore Default layout	Restore the default IntelliJ IDEA layout (tool windows positions, buttons location and order). To restore the default layout, check the option Store Current Layout as Default in the Window menu.

Ctrl+Alt+Shift

This section lists and describes the keyboard shortcuts that include the `Ctrl+Shift+Alt` keys.

Shortcut	Function	Use this shortcut to...
`Ctrl+Shift+Alt+C`	Copy Reference	Copy a reference of a symbol to the Clipboard.
`Ctrl+Shift+Alt+N`	Go to Symbol	Navigate to a symbol with the specified name.
`Ctrl+Shift+Alt+H`	Pop up Hector	Open the Highlighting level pop-up window.
`Ctrl+Shift+Alt+S`	Project Structure	Open Project Structure dialog box.
`Ctrl+Shift+Alt+U`	Show Uml Diagram	Open UML Class diagram for a class or package.
`Ctrl+Shift+Alt+V`	Paste Simple	Paste the last entry from the Clipboard as plain text.
`Ctrl+Shift+Alt+L`	Show Reformat File Dialog	Show reformatting dialog .
`Ctrl+Shift+Alt+I`	Run Inspection by Name	Execute an inspection by its name .
`Ctrl+Shift+Alt+Insert`	New Scratch File	Create a new scratch file with the selected language.

Keyboard Shortcuts By Category

In this part you can find reference information about the keyboard shortcuts grouped by functional categories:

- Advanced Editing
- Basic Editing
- Code Folding
- Running and Debugging
- General
- GUI Designer Shortcuts
- Search
- Navigation Between Bookmarks
- Navigation Between IDE Components
- Navigation in Source Code
- Refactoring

Advanced Editing

Function	Shortcut	Use this shortcut to...
Comment with Line Comment	`Ctrl+Slash`	Comment/uncomment current line or selected block with line comments.
Comment with Block Comment	`Ctrl+Shift+Slash`	Comment/uncomment code with block comments.
Quick Documentation	`Ctrl+Q` / `Alt+Button2 Click`	Show a pop-up window with the documentation for the symbol at the caret.
Show Table Data	`Ctrl+Q`	Show a pop-up window that displays the `create table` query for the database table at the caret and the first 10 rows of the table.
Pop-up Hector	`Ctrl+Shift+Alt+H`	Show the Highlighting level pop-up window to configure highlighting in the current file.
Parameter Info	`Ctrl+P`	Show parameters of the method call at the caret.
Context Info	`Alt+Q`	Show the current method or class declaration when it is not visible.
Error Description	`Ctrl+F1`	Show an error or warning description at the caret.
External Documentation	`Shift+F1`	Open browser with the documentation for the selected item.
Override Methods...	`Ctrl+O`	Override base class methods in the current class.
Implement Methods...	`Ctrl+I`	Override base interface/class methods in the current class.
Surround with...	`Ctrl+Alt+T`	Surround selected code fragment with `if`, `do`, tags or other constructs. In the GUI Designer, use this shortcut to wrap selected components into a container.
Generate	`Alt+Insert`	Generate constructors, accessor methods, EJB components, Maven dependencies in the `pom.xml` files, using the Generate pop-up menu.
Basic Code Completion	`Ctrl+Space` `Alt+Slash`	Code completion for any class, method or variable.
SmartType Code Completion	`Ctrl+Shift+Space`	Code Completion filtering the lookup list basing on expected type.
Expand Word	`Alt+Slash`	Goes through the names of classes, methods, keywords and variables in the current visibility scope.
Insert Live Template...	`Ctrl+J`	Show a pop-up list of starting with a specified prefix.
Surround with Live Template...	`Ctrl+Alt+J`	Surround the selection with one of the templates.
Next Template Variable	`Tab`	In templates: move the caret to the next template variable.
Previous Template Variable	`Shift+Tab`	In templates: move the caret to the previous template variable.

Basic Editing

Function	Shortcut	Use this shortcut to...
Enter	Enter	Depending on the context: – In a lookup list: select an item. – In the editor: enter a new line and set the caret at its beginning.
Tab	Tab	In the editor: – With selection: indent selected lines. – Without selection: insert a tab symbol (or corresponding number of space characters). In a lookup list: – No code after the caret: select an item. – Some code after the caret: select an item and substitute the code after the caret with it.
Delete	Delete	Depending on the context: – In the editor: delete selected symbol/block. – In a usage view: exclude a selected item. – In other views: remove selected items.
Backspace	Backspace	Delete a character to the left of the caret.
Undo	Ctrl+Z	Undo last operation.
Redo	Ctrl+Shift+Z	Redo last undone operation.
Cut	Ctrl+X	Cut a current line or a selected code block to the Clipboard.
Copy	Ctrl+C	Copy a current line or a selected code block to the Clipboard.
Paste	Ctrl+V	Paste from the Clipboard to the caret location.
Paste from History	Ctrl+Shift+V	Paste selected entry from the Clipboard to the caret location.
Up	Up	Move the caret one line up.
Up with Selection	Shift+Up	Move the caret one line up selecting the text.
Down	Down	Move the caret one line down.
Down with Selection	Shift+Down	Move the caret one line down selecting the text.
Left	Left	Move the caret one character to the left.
Left with Selection	Shift+Left	Move the caret one character to the left selecting the text.
Right	Right	Move the caret one character to the right.
Right with Selection	Shift+Right	Move the caret one character to the right selecting the text.
Go to Page Bottom	Ctrl+Page Down	Move the caret down to the page bottom.
Go to Page Bottom with Selection	Ctrl+Shift+Page Down	Move the caret down to the page bottom, selecting the text.
Go to Page Top	Ctrl+Page Up	Move the caret up to the page top.
Go to Page Top with Selection	Ctrl+Shift+Page Up	Move the caret up to the page bottom, selecting the text.
Page Down	Page Down	Move the caret one page down.
Page Down with Selection	Shift+Page Down	Move the caret one page down, selecting the text.
Page Up	Page Up	Move the caret one page up.
Page Up with Selection	Shift+Page Up	Move the caret one page up, selecting the text.
Scroll Down	Ctrl+Down	Scroll the text one line down.
Scroll to Center	Ctrl+M	Scroll a line at caret to the center of the screen.
Scroll Up	Ctrl+Up	Scroll the text one line up.
Move to Line End	End	Move the caret to the end of line.
Move to Line End with Selection	Shift+End	Move the caret to the end of line, selecting the text.
Move to Line Start	Home	Move the caret to the beginning of line.
Move to Line Start with Selection	Shift+Home	Move the caret to the beginning of line, selecting the text.
Move to Next Word	Ctrl+Right	Move the caret to the next word.
Move to Next Word with Selection	Ctrl+Shift+Right	Move the caret to the next word, selecting it.
Move to Previous Word	Ctrl+Left	Move the caret to the previous word.
Move to Previous Word with Selection	Ctrl+Shift+Left	Move the caret to the previous word, selecting it.
Move to Text End	Ctrl+End	Move the caret to the end of text.
Move to Text End with Selection	Ctrl+Shift+End	Move the caret to the end of text, selecting it.
Move to Text Start	Ctrl+Home	Move the caret to the beginning of text.
Move to Text Start with Selection.	Ctrl+Shift+Home	Move the caret to the beginning of text, selecting it.
Select All	Ctrl+A	Select the entire text opened in the editor.

Delete Line at Caret	Ctrl+Y	Delete the line where the caret is currently located.
Delete to Word End	Ctrl+Delete	Delete the word starting from the current caret location up to the word end.
Delete to Word Start	Ctrl+Backspace	Delete the word starting from the current caret location up to the word start.
Toggle Insert/Overwrite	Insert	Toggle insert/overwrite modes.
Duplicate Line or Block	Ctrl+D	Duplicate selected block or the line at the caret.
Toggle Case	Ctrl+Shift+U	Toggle case of the selected text block.
Move to Code Block End	Ctrl+Close Bracket	Move the caret to the current code block end, highlighting the block limits.
Move to Code Block End with Selection	Ctrl+Shift+Close Bracket	Move the caret to the current code block end, selecting the code beginning from the initial caret location.
Move to Code Block Start	Ctrl+Open Bracket	Move the caret to the current code block start, highlighting the block limits.
Move to Code Block Start with Selection	Ctrl+Shift+Open Bracket	Move the caret to the current code block start, selecting the code beginning from the initial caret location.
Start New Line	Shift+Enter	Start a new line after the current one positioning the caret in accordance with the current indentation level.
Start New Line Before Current One	Ctrl+Alt+Enter	Start a new line before the current one.
Join Lines	Ctrl+Shift+J	Concatenate the selected lines into one or concatenate the line where the caret is currently located with the next line.
Split Line	Ctrl+Enter	Split the selected line at the point where the caret is located, leaving the caret at the end of the first line.
Select Word at Caret	Ctrl+W	Select successively increasing code blocks starting from the current caret location.
Unselect Word at Caret	Ctrl+Shift+W	Remove sequentially the selection made by the action.
Indent Selection	Tab	Move the selected block to the next indentation level.
Unindent Selection	Shift+Tab	Move the selected block to the previous indentation level.
Auto-Indent Lines	Ctrl+Alt+I	Indent the current line or selected block according to the Code Style settings.

Code Folding

Command	Shortcut	Description
Expand	`Ctrl+NumPad Plus`	Expand the current collapsed fragment
Collapse	`Ctrl+NumPad -`	Collapse the current folding region
Expand Recursively	`Ctrl+Alt+NumPad Plus`	Expand the current folded fragment and all the subordinate collapsed folding regions within that fragment
Collapse Recursively	`Ctrl+Alt+NumPad -`	Collapse the current folding region and all the subordinate folding regions within it
Expand All	`Ctrl+Shift+NumPad Plus`	Expand all collapsed fragments within the selection, or, if nothing is selected, expand all the collapsed fragments in the current file
Collapse All	`Ctrl+Shift+NumPad -`	Collapse all folding regions within the selection, or, if nothing is selected, collapse all the folding regions in the current file

Expand to level | 1, 2, 3, 4 or 5	`Ctrl+NumPad *, 1`
	`Ctrl+NumPad *, 2`
	`Ctrl+NumPad *, 3`
	`Ctrl+NumPad *, 4`
	`Ctrl+NumPad *, 5`
		Expand the current fragment and all the nested fragments up to the specified level

Expand all to level | 1, 2, 3, 4 or 5	`Ctrl+Shift+NumPad *, 1`
	`Ctrl+Shift+NumPad *, 2`
	`Ctrl+Shift+NumPad *, 3`
	`Ctrl+Shift+NumPad *, 4`
	`Ctrl+Shift+NumPad *, 5`
		Expand all the collapsed fragments in the file up to the specified nesting level

Fold Selection / Remove region	`Ctrl+Period`	Collapse the selected fragment and create a custom folding region for it to make it "foldable" / Expand the current fragment and remove the corresponding custom folding region to make the fragment "unfoldable"
Fold Code Block	`Ctrl+Shift+Period`	Collapse the code fragment between the matched pair of curly braces `{}` and create a custom folding region for that fragment to make it "foldable"

Running and Debugging

Function	Shortcut	Use this shortcut to...
Make Project	Ctrl+F9	Compile all modified and dependent files in a project.
Compile	Ctrl+Shift+F9	Compile selected file/package.
Run	Shift+F10	Run a program.
Choose configuration and run	Shift+Alt+F10	Quickly select run/debug configuration and run or edit it.
Rerun	Ctrl+F5	Repeat execution with the same settings, with the same tab of the Run tool window having the focus.
Rerun without loosing the focus in the editor	Shift+F10	Repeat execution with the same settings, with the same tab of the editor having the focus.
Debug	Shift+F9	Debug a program.
Choose configuration and debug	Shift+Alt+F9	Quickly select run/debug configuration and debug or edit it.
Update running application	Ctrl+F10	Reload updated resources and recompiled classes of the currently running application.
Step Over	F8	Step to the next line in the current file. See Stepping Through the Program .
Step Into	F7	Step to the next executed line. See Stepping Through the Program .
Smart Step Into	Shift+F7	Select the method to step in, if the current line contains multiple method call expressions. See Choosing a Method to Step Into .
Step Out	Shift+F8	Step to a first executed line after returning from the current method. See Stepping Through the Program .
Force Step Over	Shift+Alt+F8	Run until the next line in this method or file, skipping the methods referenced at the current execution point and ignoring breakpoints. See Stepping Through the Program .
Force Step Into	Shift+Alt+F7	Steps into the method called in the current execution point even if this method is to be skipped. See Stepping Through the Program .
Run to Cursor	Alt+F9	Run to the line where the caret is located. See Stepping Through the Program .
Force Run To Cursor	Ctrl+Alt+F9	Run to the line where the caret is located, ignoring breakpoints. See Stepping Through the Program .
Resume Program	F9	Resume program execution.
Stop Program	Shift+F2	Terminate a debugging session.
Evaluate Expression	Alt+F8	Evaluate an arbitrary expression.
Quick Evaluate Expression	Ctrl+Alt+F8	Evaluate an arbitrary expression without calling Evaluate Expression dialog.
Toggle Breakpoint	Ctrl+F8	Toggle breakpoint at the current line.
View Breakpoints	Ctrl+Shift+F8	View/manage all breakpoints.
Switch to another coverage suite.	Ctrl+Alt+F6	Open the Coverage Suites pop-up window and select the desired suite to run.

General

Function	Shortcut	Use this shortcut to...
Close Active Tab	Ctrl+Shift+F4	Close an active tab in a tool window (for example, Find tool window).
Close Editor	Ctrl+F4	Close an active editor.
Edit Source	F4	Open an editor for the selected item or items and give focus to the last opened file.
Escape	Escape	Depending on the context: – In the editor: close pop-up windows, terminate search, or remove highlighting. – In a tool window: return focus to the editor.
Export to Text File	Alt+O	Export contents of a tool window to a text file. This feature applies to the Version Control Tool Window , Messages Tool Window , and other tool windows that provide the export button on the window toolbar.
New...	Alt+Insert	Create a new class, interface, file or directory. See Populating Modules .
Save All	Ctrl+S	Save all files and settings.
Select Next Tab	Alt+Right	When several tabs are open in the editor or a view, open the next tab to the right (or first tab if the current one is the last).
Select Previous Tab	Alt+Left	When several tabs are open in the editor or a view, open the next tab to the left (or last tab if the current one is the first).
Show Intention Action	Alt+Enter	Display intention actions (if any) for the code where the caret is currently located, or the selected GUI component in a form.
Synchronize	Ctrl+Alt+Y	Detect all externally changed files and reload them from disk.
View Source	Ctrl+Enter	Depending on the context: – In Tool Windows: Open an Editor tab or tabs for the selected item or items, respectively. – In the editor: Intelligently split the current line into 2 lines, shifting quotes, etc. as necessary. – On the context menus of the modules in the Project tool window , Dependency Viewer, and Module Dependencies tool window : open the Modules structure .

GUI Designer Shortcuts

Function	Shortcut	Use this shortcut to...
Select next component	`Arrow`	Move selection to the adjacent component.
Add to selection	`Shift+Arrow`	Add adjacent component to the selection.
Extend selection	`Ctrl+W` `Ctrl+Shift+W`	Select successively increasing sets of components from the current component to its container. Compare to selecting text in the editor .
Move component	`Ctrl+Arrow`	Move selected component to the adjacent valid container.
Expand component	`Ctrl+Shift+Arrow`	Expand component to the adjacent valid container.

Search

Function	Shortcut	Use this shortcut to...
Find	Ctrl+F	Initiate text search .
Replace	Ctrl+R	Initiate text search and replace .
Search for next/ previous occurrence	F3 / Shift+F3	Navigate to the next/previous occurrence of a selected word in the editor.
Find Word at Caret	Ctrl+F3	Search in the editor for the word where the caret is currently located.
Incremental Search	Ctrl+F	Initiate text search .
Find in Path	Ctrl+Shift+F	Initiate search for a text string in the specified scope .
Replace in Path	Ctrl+Shift+R	Initiate search and replace in the specified scope .
Find Usages	Alt+F7	Initiate search for usages of the selected symbol in the specified scope .
Find Usages in File	Ctrl+F7	Initiate search for usages of the selected symbol in the current file .
Highlight Usages in File	Ctrl+Shift+F7	Highlight usages of a symbol at caret.
Show Usages	Ctrl+Alt+F7	Show usages of a symbol at caret in a pop-up window. Use list of found usages to jump to the desired location.
Find Action	Ctrl+Shift+A	Find an action, bypassing menus. See Finding Actions .

Navigation Between Bookmarks

Function	Shortcut	Use this shortcut to...
Go to Bookmark <number>	Ctrl+Number	Navigate to a numbered bookmark with the corresponding number.
Toggle Bookmark	F11	Turn anonymous bookmark on or off.
Toggle Bookmark with Mnemonic	Ctrl+F11	Turn bookmark with mnemonic on or off.
Show Bookmarks	Shift+F11	Open Bookmarks dialog to manage existing bookmarks and navigate between them.

Navigation Between IDE Components

In this section you can find keyboard shortcuts for navigation between:

– Views and Windows .

– Differences .

Views and Windows

Function	Shortcut	Use this shortcut to...
Select Target	Alt+F1	Move focus from the current file, class, method or reference to a data source, to a view suggested in the Select Target pop-up menu. Refer to Navigating Between IDE Components .
Collapse all	Ctrl+NumPad -	Collapse all nodes in a tree view.
Expand all	Ctrl+NumPad Plus	Expand all nodes in a tree view.
Switcher	Ctrl+Tab	Navigate between files opened in the editor, and tool windows.
Open tool window	Alt+Number	Open a tool window with the specified number.
Hide Active Window	Shift+Escape	Hide the currently active tool window.
Jump to Last Window	F12	Activate the last focused tool window.

Differences

Function	Shortcut	Use this shortcut to...
Move to Next Difference	F7	Navigate to the next difference in view .
Move to Previous Difference	Shift+F7	Navigate to the previous difference in view .

Navigation In Source Code

Function	Shortcut	Use this shortcut to...
File Structure Pop-up	Ctrl+F12	Display the Structure pop-up window for quick navigation through the current file.
Select target	Alt+F1	Move focus from the current file, class, method or reference to a data source table to a view suggested in the Select Target pop-up menu. See Navigating Between IDE Components .
Recent Files	Ctrl+E	Show the list of recently opened files .
Recently Changed Files	Ctrl+Shift+E	Show the list of recently updated files .
Type Hierarchy	Ctrl+H	Browse hierarchy for the selected class .
Method Hierarchy	Ctrl+Shift+H	Browse hierarchy for the selected method .
Call Hierarchy	Ctrl+Alt+H	Browse call hierarchy for the selected method.
Navigate to Class	Ctrl+N	Navigate directly to a class in project by specifying its name in a pop-up dialog box.
Navigate to File	Ctrl+Shift+N	Navigate directly to a file in project by specifying its name in a pop-up dialog box.
Navigate to Recently Opened File	Ctrl+E	Show the list of recently opened files .
Navigate to Recently Changed File	Ctrl+Shift+E	Show the list of recently updated files .
Navigate to Line	Ctrl+G	Navigate to any line in the current file by specifying its number.
Navigate to Declaration	Ctrl+B	Navigate to declaration of a symbol at caret.
Navigate to Implementation	Ctrl+Alt+B	Navigate to implementation of the item at caret.
Navigate to Type Declaration	Ctrl+Shift+B	Navigate to a type declaration of a symbol at caret, the symbol being a variable or a method call.
Navigate to Super Method	Ctrl+U	Navigate to a super method declaration of a method under the caret.
Navigate to Test/Test Subject	Ctrl+Shift+T	Navigate to a test for the class at caret, if any, or navigate from a test to a test subject.
Navigate to Related Symbol	Ctrl+Alt+Home	Navigate between files with complicated relationships between them. For example, use this shortcut to navigate between the various web entities.
Navigate to Next Method	Alt+Down	Navigate to the next method declaration in the active editor tab.
Navigate to Previous Method	Alt+Up	Navigate to the previous method declaration in the active editor tab.
Navigate to Opening Brace	Ctrl+Open Bracket	Navigate to the start of the current code block.
Navigate to Closing Brace	Ctrl+Close Bracket	Navigate to the end of the current code block.
Back	Ctrl+Alt+Left	Undo last navigation operation. Note On an OS X computer, you can also use the three-finger right-to-left swipe gesture.
Forward	Ctrl+Alt+Right	Redo last undone navigation operation. Note On an OS X computer, you can also use the three-finger left-to-right swipe gesture.
Navigate to Previous Occurrence	Ctrl+Alt+Up	Navigate to a previous found item.
Navigate to Next Occurrence	Ctrl+Alt+Down	Navigate to a next found item.
Last Edit Location	Ctrl+Shift+Backspace	Move through the most recent change points.
Navigate to Next Highlighted Error	F2	Navigate to the next found error/warning.
Navigate to Previous Highlighted Error	Shift+F2	Navigate to the previous found error/warning.

Refactoring

Function	Shortcut	Use this shortcut to...
Rename	`Shift+F6`	Rename the selected file, class, field, method, etc. and change all references to it accordingly.
Change Method Signature	`Ctrl+F6`	Change the signature of the selected method and update all the corresponding method calls.
Move	`F6`	Move the selected class, package or static member to another package or class and update all the corresponding references.
Copy	`F5`	Create a copy of the selected class, file or directory in the same or different directory or package.
Clone		Create a copy of the selected class in the same package.
Safe Delete	`Alt+Delete`	Delete the selected class, method or field checking its usages.
Extract Method	`Ctrl+Alt+M`	Turn the selected code fragment into a method.
Extract Variable	`Ctrl+Alt+V`	Create a new variable and use the selected expression as its value.
Extract Field	`Ctrl+Alt+F`	Create a new field and use the selected expression as its value.
Extract Constant	`Ctrl+Alt+C`	Create a new constant (static final field) and use the selected expression as its value.
Extract Parameter	`Ctrl+Alt+P`	Turn the selected expression into a new method parameter.
Inline	`Ctrl+Alt+N`	Inline the selected method or variable.

Mouse Reference

Item	Description
Middle mouse button rotate	Scroll vertically
`Shift` + Middle mouse button rotate	Scroll horizontally
`Ctrl` + Middle mouse button rotate	Change font size
Middle mouse button click on a tab	Close tab
Right mouse button click	Show context menu

Version Control Reference

This part provides miscellaneous information, related to common version control operations, and to VCS integrations:

- CVS Reference
- Git Reference
- Mercurial Reference
- Perforce Reference
- Subversion Reference
- Apply Patch Dialog
- Checkout from TFS Wizard
- Create Patch Dialog
- Commit Changes Dialog
- Configure Ignored Files Dialog
- Enable Version Control Integration Dialog
- File Status Highlights
- New Changelist Dialog
- Patch File Settings Dialog
- Revert Changes Dialog
- Select Target Changelist Dialog
- Shelve Changes Dialog
- Unshelve Changes Dialog
- Visual SourceSafe Options Dialog

CVS Reference

In this part:

- CVS Global Settings Dialog
- CVS Options Dialog
- CVS Root Dialog
- CVS Tool Window

CVS Global Settings Dialog

File | Settings | Version Control | VCSs | CVS for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | VCSs | CVS for OS X

`Ctrl+Alt+S`

Use this dialog box to set up CVS options at the global level. The dialog is available for files and directories that are under CVS control.

Item	Description
Charset	From this drop-down list, select the character set to be used.
Use gzip compression	Select this check box to apply `gzip` compression.
Password file	In this text box, specify the fully qualified path to the `.cvspass` file. Click ... to select the file in the corresponding dialog .
Connection timeout	In this text box, type the timeout value in seconds.
Send environment variable to server	Select this check box to have CVS-related environment variables sent to the server.
Log CVS client/server output to `cvs.log` file	Select this check box to enable logging and have the `cvs.log` log file stored in the `log` directory of your IntelliJ IDEA installation.

CVS Roots Dialog

VCS | CVS | Configure CVS Roots

Use this dialog box to set up CVS roots. The dialog box is available for files and directories that are under CVS version control.

Common Options

Item	Description
+	Click this button to configure a new CVS root.
−	Click this button to remove the selected CVS root configuration from the list.
	Click this button to create a copy of the selected CVS root.
Global Settings	Click this button to open the Global CVS Settings dialog box where you can set up CVS options at the global level.
CVS root	In this text box, specify the CVS repository string according to the following syntax: `[:method:][[user][:password]@]hostname[:[port]]/path/to/repository` . Tip Obtain the valid string from your system administrator or click the Edit by Field button to open the Configure CVS Root Field by Field dialog box where you can specify the mandatory connection parameters and have IntelliJ IDEA assemble them into a correct repository string.
Edit by Field	Click this button to open the Configure CVS Root Field by Field dialog box where you can specify the mandatory connection parameters and have them assembled into a CVS root string automatically.
Use version	Use this section to specify the revision you want to synchronize your local working copy with. The available options are: – HEAD revision: this option is suggested by default. – By tag: select this option to access the revision with a specific tag. Type the desired tag in the text box or click the Browse button … and select the desired tag from the list. The list shows all the tags available on the CVS server according to the specified CVS root. – By date: select this option to access the revision with a specific date and time stamp. Type the end date and time in the format `dd:mm:yy hh:mm:ss` or click the Browse button … and select the desired date from the calendar. This date and time are passed to the server with the GMT parameter. Tip The controls in the area are available only after the CVS root text box is filled in with valid data. Warning! If you perform update or checkout from the CVS repository with the By tag or By date option selected, the resulting working copy will be permanently restricted to the specified tag or date, until you force the update operation to reset this sticky data .
Test connection	Click this button to check that the specified settings ensure successful connection to the CVS server.

Additional Connection Settings

In this area, specify additional settings to flexibly configure connection to the CVS server. The contents of the area depends on the connection method set in the CVS root text box.

– pserver

– ext

– ssh

– local

Pserver

Warning! The settings specified in this area affect all CVS roots that use the pserver connection method.

Item	Description
Password	In this text box, type the fully qualified path to the `.cvspass` file. Click the Browse button … to select the file in the corresponding dialog .
Connection timeout	In this text box, type the connection timeout in seconds.
Proxy Settings	See the Proxy Settings section below.

Ext

Item	Description
Use internal `ssh` implementation	Select this check box to access the ssh area , where you can specify the SSH version to use, the port to listen to, and configure your private key and password. Clear this check box to access the Ext Protocol Settings area with the following controls available: – Path to external rsh: in this text box, specify the location of the external `rsh` . If necessary, click the Browse button … to select the necessary location in the corresponding dialog . – Path to private key file: in this text box, specify the location of the file with your private `ssh` key. If necessary, click the Browse button … to select the file in the corresponding dialog . – Additional parameters: in this text box, specify additional connection parameters.

Ssh

This area is also available when you have specified the ext connection method and selected the Use internal ssh implementation check box.

Item	Description
SSH version	In this area, specify the SSH version to use. The available options are: – Allow both – Force SSH1

– Force SSH2

Port	In this text box, specify the `ssh` port to listen to.
Use Private key file	Select this check box, if you want to pass server authentication using a private `ssh` key. In the text box, specify the location of the file with your private `ssh` key. If necessary, click the Browse button … to select the file in the corresponding dialog .
Change password	Click this button to open the SSH PAssword dialog box, where you can specify the password for the curent CVS root.
Proxy Settings	See Proxy Settings section below.

Local

Tip IntelliJ IDEA does not provide the server functionality. If you want to use a local CVS client, you need to install CVS on your local host computer and configure it to work as a server.

Item	Description
Path to CVS client	In this text box, type the path to CVS client installed on the host computer and configured to work as a server. If necessary, click the Browse button … to select the necessary location in the corresponding dialog .
Server command	In this text box, specify the server command.

Proxy Settings

Item	Description
Use proxy	Select this check box to enable using the Proxy server and access the Login, Password, Proxy host, and Proxy port text boxes below. This check box is available only in two cases: – The connection method is `pserver` or `ssh` . – The connection method is `ext` and the Use internal ssh implementation check box is selected.
Protocol	Select the protocol to use. The available options are: – HTTP – Socks4 – Socks5
Login	In this text box, specify your user name.
Password	In this text box, specify the user password.
Proxy host	In this text box, specify the Proxy host name.
Proxy port	In this text box, specify the Proxy port number.

CVS Tool Window

VCS | Browse CVS Repository

View | Tool Windows | CVS

This tool window opens, when you browse a CVS repository, and enables you to view contents of the repository, check out files, browse changes, view annotations and navigate to source code.

Item	Description
	Click this button to close the current tab.
	Click this button to open the selected file in the editor.
	Click this button to obtain a local copy of the selected file or directory.
	Click this button to open selected file in the editor with the annotations turned on. See Viewing Annotations .
	Click this button to see the changes that affect the selected file or directory, and that have been committed to the repository by a certain user, or during the specified period. The filtering information is entered in the Search Criteria dialog. Search results show in a dedicated tab of the Version Control tool window .
	Click this button to show reference page.

Import into CVS

VCS | Import into CVS

Use this dialog to import a directory into the specified CVS repository.

Select CVS Configuration

Use this page to select the target CVS root and change its configuration , if necessary.

Select Directory to Import to

Use this page to select the target directory.

Select Import Directory

Use this page to select the directory to be imported. If you are importing an IntelliJ IDEA project, make sure that the project file is located under that directory. Multiple selection is not available.

Customize Keyboard Substitution

Use this page to specify the keyword substitution rule for the files imported into the repository.

Import Settings

Item	Description
Name in repository	In this field, specify the name that corresponds to the `module` argument. Tip For import, `module` refers to the absolute location in the repository, not to a module name defined in the modules file.
Vendor	In this field, specify the name that corresponds to the `vendor-tag` argument. This tag is used as a branch tag. No checkouts will ever be done explicitly on it. Type a name that is relevant to the project, or just VENDOR.
Release tag	In this text box, specify the string that corresponds to the `release-tag` argument. The tag should refer to a version or a release number. Tip A tag name cannot contain punctuation marks. For example: `-release-2.2` is wrong `release-2-2` is correct.
Log message	In this field. specify the string that corresponds to the `-m` command-line argument. By default, the field shows the previous log message; you can accept default, or type a new comment.
Checkout after import	Select this option to have CVS checkout run after completing the import operation.
Make checked out files read-only	Select this option to mark the checked out files as read-only after import. This option is disabled if the Checkout after import option is cleared.

Check Out From CVS Dialog

VCS | Checkout From Version Control | CVS

The dialog consists of the following pages:

- Select CS Configuration
- Select CVS Element to Check Out
- Select Checkout Location
- Check out to

Select CVS Configuration

Item	Description
List of available CVS configurations	Use this list to select the desired CVS configuration.
Configure	Click this button to define a new CVS configuration, or modify an existing one, in the CVS Roots dialog box.

Select CVS Element to Check Out

Use this page to select elements of the repository to check out. Next button is only available when an element is selected.

Select Checkout Location

Use this page to specify the target location for the artifacts to check out. All actions can be performed using the toolbar buttons, or context menu.

Item	Shortcut	Description
	Ctrl+1	Navigate to the user home.
	Ctrl+2	Navigate to the project root directory.
	Alt+Insert	Create a new directory, where the files will be checked out to.
	Delete	Delete selected directory.
	Ctrl+Alt+Y	Synchronize with external changes.

Check out to

Use this page to define CVS-specific checkout options.

Item	Description
List of local paths	Select the local path to which the module name should be added.
Make new files read-only	Check this option to set read-only attribute for the files that did not exist locally but were checked out from the repository.
Prune empty directories	Check this option to delete empty directories from the repository.
Change keyword substitution to	Check this option to enable keyword substitution, and select the desired substitution mode from the drop-down list.

Configure CVS Root Field by Field Dialog

VCS | CVS | Configure CVS Roots

This dialog box opens when you click the Edit by Field button in the CVS Roots dialog box. Use this dialog box to specify the parameters for connecting to the CVS server and have IntelliJ IDEA assemble them into a correct repository string according to the CVS root string syntax.

Item	Description
Method	From this drop-down list, select the desired connection method. The available options are: – pserver – ext – ssh (internal implementation) – local
User	In this text box, type your login to the CVS server.
Port	In this text box, specify the port to listen to on the CVS server host.
Host	In this text box, type the name of the host where the desired CVS server is located.
Repository	In this text box, type the path to the CVS repository relative to the host name.

Rollback Actions With Regards to File Status

In this section:

- Menu commands according to file status .
- Effect of rolling back local changes .

Menu Commands According to File Status

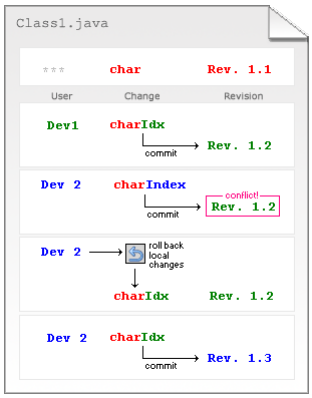
Depending on the file status, the Rollback Changes command will be *aliased* as shown in the following table:

File status	Rollback Command	Result
`modified`	Rollback Local Changes	all changes made in the file will be reverted, and the file will acquire the *up to date* status
`deleted`	Rollback Deletion	file will be restored both on the disk and in CVS with the *up to date* status
`externally deleted`	Rollback Deletion	file will be restored on the disk and assigned the *up to date* status
`added`	Rollback Creation	file will be deleted from disk
`merged`	Rollback Local Changes	all local changes will be dropped, changes from the repository will be accepted, and the file will be assigned the *up to date* status
`merged with conflicts`	Rollback Local Changes	all local changes will be dropped, changes from the repository will be accepted, and the file will be assigned the *up to date* status
`unknown`	Rollback Creation	file will be deleted from the disk

Effect of rolling back local changes

The effect of Rollback Local Changes may not be what you intuitively expect in terms of the revision you have locally after running the command.

The image below represents a file in CVS and a sequence of actions by two developers. It shows a simple example of what happens in terms of the local copy's CVS revision after rolling back conflicting local changes.



Here's what happens:

- The developer *Dev1* takes Revision 1.1 from the repository, modifies it, and commits changes to CVS.
- The developer *Dev2* doesn't know about *Dev1*'s changes and modifies the same code in the local copy of Revision 1.1. When *Dev2* commits these changes, he gets a message from CVS that the repository has changed. So he runs Update to synchronize, and his local copy is then updated to Revision 1.2, and CVS sets the *Merged with Conflicts* status on the file.
- *Dev2* decides to roll back local changes. He is left with a local copy of Revision 1.2 .
- When *Dev2* commits the file to CVS, it becomes Revision 1.3 even though the content is identical to Revision 1.2.

Update Directory / Update File Dialog (CVS)

VCS | CVS | Update File

VCS | CVS | Update Directory Editor | context menu of a file | CVS | Update File

Project Tool Window | context menu of a file | CVS | Update File

Project Tool Window | context menu of a selection | CVS | Update Files

Project Tool Window | context menu of a folder | CVS | Update Directory

Project Tool Window | context menu of a selection | CVS | Update Directories

---

In these dialog boxes, configure synchronization of local files or folders with the repository.

Item	Description
Branch Merging	In this area, specify the repository branch to synchronize with. – Don't merge - select this option to synchronize with the counterpart of the current branch in the repository. This option is selected by default. – Merge with branch - select this option to have the local copy synchronized with a repository branch different from the counterpart of the current branch. In the first text box below, specify the branch to synchronize with. Type the branch name manually or click the Browse button and select the desired branch from the list in the Select Tag dialog bos that opens. This option is equivalent to the `-j` command-line option of the `update` command. – Merge two branches - select this option to have the local copy synchronized with the result of merging two repository branches different from the counterpart of the current branch. In the text boxes below, specify the branches to synchronize with. This option is equivalent to the `-j -j` command-line option of the `update` command.
Use Version	In this area, specify the version of repository file(s) or folder(s) to synchronize with. – Default - select this option to have the local copy synchronized with the latest repository version. – By tag - select this option to have the local copy synchronized with a particular repository version. In the text box, specify the desired version number or tag. Type the version number or tag manually or click the Browse button and select the desired branch from the list in the Select Revision or Tag dialog bos that opens. – When updating a single file, you can specify its repository counterpart either through a version number or a tag. – When updating an entire folder, its repository counterpart can be specified only through a tag. This option is equivalent to the `-r` command-line option of the `update` command. – By date - select this option to have the local copy synchronized with a repository version that was submitted on a particular date. Type the date manually or click the Calendar button and select the desired date from the calendar pop-up window that opens with the current date selected by default. This option is equivalent to the `-D` command-line option of the `update` command.
Reset sticky data	Select this check box to remove the date or tag restriction from the local file(s) or folder(s) to be updated. Such restrictions are set on files and folders checked out or previously updated with the By tag or By date options. This option is equivalent to the `-A` command-line option of the `update` command.
Prune empty directories	Select this option while updating a directory to have IntelliJ IDEA remove the subfolders whose repository counterparts are empty. This option is equivalent to the `-P` command-line option of the `update` command.
Change keyword substitution to	Select this check box to have the default keyword expansion mode changed. From the drop-down list, choose the relevant substitution. This option is equivalent to the `-k` command-line option of the `update` command.
Create new directories	Select this option while updating a directory to have IntelliJ IDEA create new local subfolders when any new subfolders have been created in the repository counterpart of the directory to be updated. This option is equivalent to the `-k` command-line option of the `update` command. By default, all the new subfolders will be picked including empty ones. To avoid creating empty subfolders, it is recommended that you select the Prune empty directories check box as well.
Clean copy	Select this check box to have IntelliJ IDEA backup the local changes and replace the changed file(s) with their counterparts from the repository. This option is equivalent to the `-C` command-line option of the `update` command.
Do not show this dialog in the future	Select this option to have the update operation performed silently in the future. To have IntelliJ IDEA show this dialog box before update again: 1. Open the Version Control - Confirmation page of the Settings dialog box. 2. In the Display Option dialogs when these commands are invoked area, select the Update check box.

Git Reference

In this part:

- Checkout Dialog
- Clone Repository Dialog
- GitHub Integration Reference
- Line Separators Warning Dialog
- Merge Branches Dialog
- Pull Changes Dialog
- Git Push Dialog
- Push Rejected Dialog (Git)
- Rebase Branches Dialog
- Git Reset Dialog
- Reset Head Dialog
- Stash Dialog
- Tag Dialog
- Unstash Changes Dialog
- Update Project Dialog (Git)

Checkout Dialog

VCS | Git | Checkout Branch

Use this dialog box to switch between existing branches and create new branches in a local repository.

Item	Description
Git Root	From this drop-down list, select the path to the local repository in which you want to switch between existing branches or create a new branch.
Current Branch	This read-only field shows the name of the working branch in the selected local repository.
Checkout	From this drop-down list, select the branch or tag to which you want to switch or which you want to copy to a new branch. To specify a particular commit, type its commit hash or use an expression, for example, of the following structure: `<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>`. Refer to the Git commit naming conventions for details.
Validate	Click this button to check that the commit specified in the Checkout field exists and view which files were affected in it. The Validate button next to an editable field indicates that regular expressions are allowed in the field.
Include Tags	Select this check box to have tags also shown in the Checkout drop-down list.
As New Branch	In this text box, type the name of the branch to be created. You can also type the name of an existing branch and select the Override check box.
Create Ref Log	Select this check box to have a reference log for the new branch created where all changes made to the branch references will be recorded.
Track Branch	Select this check box to track the parent of the new branch.
Override	Select this check box if you want to checkout a branch to an existing branch. Use this option with care because the history of the target branch will be lost.
Checkout	Click this button to switch to the specified existing branch or to create a new branch in the local repository.

Clone Repository Dialog

Checkout from Version Control | Git

Use the dialog box to set up a local repository by downloading the data from a remote repository.

Item	Description
Git Repository URL	In this text box, type the URL of the remote repository which you want to clone.
Test	Click this button to check that connection to the remote repository has been established successfully.
Parent Directory	In this text box, specify the directory where you want IntelliJ IDEA to create a folder for your local Git repository. Type the path manually or click the Browse button and choose the desired directory in the dialog that opens .
Directory Name	In this text box, type the name of the new folder into which the repository will be cloned. Warning! The parent directory must not contain a folder with the specified name.
Clone	Click this button to start cloning the specified repository.

GitHub Integration Reference

In this part:

- Select Repository to Clone Dialog
- Share Project on GitHub Dialog
- Login to GitHub Dialog
- Create Gist Dialog
- Create Pull Request Dialog

Select Repository to Clone Dialog

VCS | Checkout from Version Control | GitHub

Use this dialog box to choose the source remote repository, the folder to clone the source to, and the name of the project to be created around the cloned data.

Item	Description
Repository	From this drop-down list, select the source repository to clone the data from.
Parent Directory	In this text box, specify the directory where the local repository for cloned sources will be created. Type the path to the directory manually or click the Browse button and choose the desired directory in the Select project destination folder dialog box that opens.
Directory Name	In this text box, specify the name of the folder where the repository will be created based on the cloned sources.
Clone	Click this button to start cloning the sources from the specified remote repository.

Share Project on GitHub Dialog

VCS | Import into Version Control | Share Project on GitHub

Use this dialog box to create a repository on the GitHub remote storage and publish your project sources there.

Item	Description
New repository name	In this text box, type the name of the repository to be created. By default, IntelliJ IDEA suggests the name of the current project.
Description	In this text box, describe the main functionality implemented in the project.
Private	Select this check box to suppress access to the repository for other users. **Warning!** The check box is disabled for free accounts.
Share	Click this button to have IntelliJ IDEA initiate repository creation and afterwards upload the project sources to the new repository.

Login to GitHub Dialog

VCS | Import into Version Control | Share Project on GitHub

Use this dialog box to log on the GitHub remote storage using your account credentials or to create a GitHub account if you do not have it yet.

Item	Description
Host	Specify the URL of your GitHub repository.
Auth Type	Use this drop-down list to select how you want to be authenticated on GitHub. − Password. If this option is selected and you have two-factor authentication enabled in your GitHub account settings, you will be asked to enter an authentication code each time IntelliJ IDEA requires you to log in to your GitHub account. − Token (recommended by GitHub for authentication from third-party applications, as it does not require IntelliJ IDEA to remember your password).
Save password	This option is only available if Password is selected as authentication method. Select this option if you want IntelliJ IDEA to remember your password.
Save token	This option is only available if Token is selected as authentication method. Select this option if you want IntelliJ IDEA to remember your GitHub authentication token.
Login	In this text box, type your GitHub logon name. This field is only available if Password is selected as authentication method above.
Password	In this text box, type your GitHub account password. This field is only available if Password is selected as authentication method above.
Token	Specify your personal access token. This field is only available if Token is selected as authentication method above.
Sign up	Click this link to open the Sign up for GitHub page where you can create a GitHub account.

Create Gist Dialog

<Context menu of an open file in the editor> | Create gist

<Context menu of a file in the Project view> | Create gist

<Context menu of a folder in the Project view> | Create gist

<Context menu of a group of files or folders selected in the Project view> | Create gist

<Context menu of a console> | Create gist

Use this dialog box to save source code and console output in gists .

Warning! The menu item and the dialog box are only available when you are logged on GitHub and a file is opened in the editor.

Item	Description
Description	In this text box, provide a brief description of the code snippet to be published.

By default, the new gist will be public, that is, visible for all registered GitHub users and your login will be shown as the gist's owner. You can change this default behaviour by selecting the check boxes below.

Private	Select this check box to make the new gist available for you only and invisible for other GitHub users.
Anonymous	Select this check box to make the new gist visible for all registered GitHub users but without displaying your login.
Open in browser	– Select this check box to have the new gist opened in the default IntelliJ IDEA browser . – To have IntelliJ IDEA display the URL address of the new gist so you can access it later, clear this check box.

Create Pull Request Dialog

VCS | Git | Create Pull Request

Creating pull requests is a nice way to share your code with the community and to follow the collaborative development. With it you can tell others about the changes you've made and ask for comments, review, or just share the knowledge. With all this going on, a pull request appears in the original repository only after approval. A pull request can be prepared right from IntelliJ IDEA without switching to the browser.

Item	Description
Base Fork	In this field, specify the repository to apply the changes to. IntelliJ IDEA attempts to retrieve all the relevant repositories and shows them in the drop-down list. Choose the repository from the drop-down list or click Select Other Fork and from the Form Owner drop-down list choose the name of the user who is the owner of the target repository.
Base Branch	In this field, specify the branch to apply the changes to. Click Show Diff to view the list of commits to be included in the pull request. To view the details of a commit, select it and switch to the Log tab which shows a list of files included in the selected commit list. See also Merging, Deleting, and Comparing Branches .
Title	In this field, specify the name of your pull request.
Description	In this field, optionally provide a brief description of the changes to be applied through the request.

Line Separators Warning Dialog

The dialog box opens when you attempt to commit a file with `CRLF` line endings, provided that you have enabled this behaviour by selecting the Warn if CRLF line separators are about to be committed checkbox in the Git page of the Settings dialog box.

Item	Description
Cancel	Click this button to stop the commit operation whereupon you can fix the problem manually.
Commit As Is	Click this button to have the file with `CRLF` line separators committed.
Fix and Commit	Click this button to have IntelliJ IDEA set the `core.autocrlf` attribute to `input`. As a result, all the `CRLF` separators will be replaced with `LF` separators and committed into the repository. Note that the reverse operation will not be performed when you download the files into your working directory, that is, no `CRLF` will appear in place of `LF`.

Merge Branches Dialog

VCS | Git | Merge Changes

Use this dialog box to specify arguments for merging branches in a local Git repository.

Item	Description
Git Root	From this drop-down list, select the path to the local repository in which you want to merge branches.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository. This is the target branch, the changes from the selected source branches will be applied to it. The contents of the field depend on the selection in the Git Root drop-down list.
Branches to Merge	Use this list box to specify the source branches from which changes will be applied to the target branch. The list shows only those branches that contain applicable commits. Applicable commits are commits made after a branch separated from the target branch.
Strategy	From this drop-down list, select the merge strategy . The available options are: – Default – Resolve - select this option if you need to resolve two HEADs, one of which is the current branch and the other HEAD is the branch which you selected in the Branches to Merge list. When this option is selected, the 3-way merge algorithm is applied. – Recursive - the default merge strategy for merging the current branch with one branch. Select this option if you need to resolve two HEADs by applying the 3-way merge algorithm and there are more than one common ancestor that can be used for 3-way merge. – Octopus - the default merge strategy for merging the current branch with more than one branch. Merges that require resolving conflicts manually are not performed. – Ours - select this option if you need to resolve several HEADs. The result of the merge is always the HEAD of the current branch. – Subtree - a modified recursive strategy. When two or more source branches are selected in the Branches to Merge list box, only the Octopus and Ours options are available.
No Commit	Select this check box if you need to inspect and, if necessary, adjust the result of merging before committing the result. The merge is performed but is not committed automatically, as if it failed.
No Fast Forward	Select this check box to generate a merge commit even if the merge resolved as a fast-forward .
Squash Commit	Select this check box to create a single commit on top of the current branch instead of merging one or more other branches. The working tree and index state are produced as if a real merge happened, but commit is not performed and the HEAD is not moved.
Add Log Information	Select this check box to have IntelliJ IDEA populate, in addition to branch names, a log message with one-line descriptions from the actual commits that are being merged.
Commit Message	In this text box, provide a description for the commit. The text box is available only if the No Commit check box is not selected.
Merge	Click this button to initiate merging the specified branches in the local repository according to the defined settings.

Pull Changes Dialog

VCS | Git | Pull Changes

Use this dialog box to specify parameters for fetching changes from a remote repository and applying them to a local repository.

Item	Description
Git Root	From this drop-down list, select the path to the local repository which you want to refresh.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository. The changes retrieved from the source remote repository will be applied to the displayed branch. The contents of the field depend on the selection in the Git Root drop-down list.
Remote	From this drop-down list, select the alias of the source remote repository. Tip To check that the selected alias corresponds to the correct URL address, expand the list - the URL addresses will be displayed explicitly.
Get Branches 	Click this button to have the Branches to Merge list box display the actual list of branches in the source remote repository.
Branches to Merge	Use this list box to specify the branches to which you want to apply the fetched changes.
Strategy	From this drop-down list, select the merge strategy . The available options are: – Default – Resolve - select this option if you need to resolve two HEADs, one of which is the current branch and the other HEAD is the branch from which you pulled changes. When this option is selected, the 3-way merge algorithm is applied. – Recursive - the default merge strategy for pulling one branch. Select this option if you need to resolve two HEADs by applying the 3-way merge algorithm and there are more than one common ancestor that can be used for 3-way merge. – Octopus - the default merge strategy for pulling more than one branch. Merges that require resolving conflicts manually are not performed. – Ours - select this option if you need to supersede old development history of side branches. By applying this strategy any number of HEADs can be resolved but the result of the merge is always the HEAD of the current branch. – Subtree - a modified recursive strategy.
No Commit	Select this check box if you need to inspect and, if necessary, adjust the result of merging the fetched data before committing the result. The merge is performed but is not committed automatically, as if it failed.
No Fast Forward	Select this check box to generate a merge commit even if the merge resolved as a fast-forward .
Squash Commit	Select this check box to create a single commit on top of the current branch instead of merging one or more other branches. The working tree and index state are produced as if a real merge happened, but commit is not performed and the HEAD is not moved.
Add Log Information	Select this check box to have IntelliJ IDEA populate, in addition to branch names, a log message with one-line descriptions from the actual commits that are being merged.
Pull	Click this button to initiate fetching changes from the specified remote repository and applying them locally according to the defined settings.

Git Push Dialog

VCS | Git | Push

In this dialog box:

- Upload the current branch to its tracked branch or to any other remote branch.
- Publish the current branch by pushing it to a new branch on the server. In this case, the local branch doesn't have a tracked branch yet and IntelliJ IDEA automatically sets up tracking (`git push -u origin newbranch`).

The dialog box consists of three panes:

- The left-hand pane shows a list of commits in the current branch . In this pane, select the commit to push changes from. To get the list of commits in the current branch up-to-date, click the Refresh commit list button .
- The right-hand pane shows the changes of the selected commit .
- In the bottom pane, choose the target branch to upload the changes to.

In this section:

- Branch Commits
- Commit Details
- Target Remote Branch

Branch Commits

This area shows a list of commits in the current branch.

- To view the details of a commit, select it in the list. The changes included in the selected commit are shown in the Commit Changes area.
- To get the list of commits in the current branch up-to-date, click the Refresh button .

Commit Details

This area shows the changes included in the selected commit. Use the controls in this area to browse through the changes included in the selected commit and appoint the changes for push by selecting the relevant files.

Item	Tooltip and shortcut	Description
	Show Diff `N/A`	Click this button to open the Differences viewer for files that shows the differences between the committed version of the selected file and its previous version thus pointing at the committed changes.
	Group by Directory `Ctrl+P`	Use this toggle button to switch between the flat view and the directory tree view.
	Expand All `Ctrl+NumPad Plus`	Click this button to have all the nodes in the directory tree unfolded. The button is not available in the flat view.
	Collapse All `Ctrl+NumPad -`	Click this button to have all the nodes in the directory tree folded. The button is not available in the flat view.
	Select All `Ctrl+A`	Click this button to select all the files in the list or directory tree and thus appoint the corresponding changes for push.

Target Remote Branch

In this area, specify the branch to upload the changes from the current branch to.

Item	Description
Push current branch to alternative branch	- Clear this check box to have the changes from the current branch uploaded to its tracked branch. - Select this check box to have the changes from the current branch uploaded to another remote branch. Choose the target branch from the drop-down list.
Target Branch	From this drop-down list, choose the target branch to upload the current branch to. If your project uses several Git repositories, the list shows only the branches that are common for all repositories. For example, if your project uses `repo_one` and `repo_two` , where `repo_one` has two branches `origin` and `github` while `repo_two` has only `origin` , then only `origin` will be available from the drop-down list, so there is no way to push to `github` from IntelliJ IDEA.
New Branch	In this text box, type the name of a new remote branch to be created, if the target branch is not yet present in the remote repository. This situation is quite common, for example, when you are publishing your project IntelliJ IDEA will automatically set up tracking (`git push -u origin newbranch`).
(`Ctrl+F5`) Refresh commit list	Click this button to get the list of commits in the current branch up-to-date.

Push Rejected Dialog (Git)

VCS | Git | Push

The dialog box opens when pushing a branch is rejected due to lack of synchronization between your local repository and the remote storage. Use this dialog box to have IntelliJ IDEA update the local branch using rebase or merge and to configure automatic update of conflicting branches in the future.

Item	Description
Update not rejected repositories as well	– Select this check box to have all the local repositories updated, no matter whether `push` has been rejected for all of them or not. – When this check box is cleared, IntelliJ IDEA will update only the repositories for which `push` was rejected. The check box is available only if your project uses several repositories.
Remember the update method choice and silently update in future	When this check box is selected, IntelliJ IDEA saves the update method `rebase` or `merge` invoked by pressing the corresponding button and will apply it to update the conflicting local branch silently. After you leave the dialog box, the Auto-update if push ... rejected check box in the Git page of the Settings dialog box is selected and the applied update method becomes default. To change this setting, clear the Auto-update if push ... rejected check box.
Merge	Click this button to have the conflicting branch updated using `merge`.
Rebase	Click this button to have the conflicting branch updated using `rebase`.

Rebase Branches Dialog

VCS | Git | Rebase

Use this dialog box to specify the branch to rebase, the new base, the rebasing mode, and configure the rebasing procedure.

Item	Description
Git Root	From this drop-down list, select the path to the local repository in which you want to rebase a branch.
Branch	From this drop-down list, select the branch to rebase. By default, the current working branch is selected. If you specify another branch, it will be automatically checked out first.
Interactive	Select this check box to view and possibly edit a list of commits to be rebased.
Preserve Merges	Select this check box to have the possibility to recreate merges instead of ignoring them. The check box is available only when the Interactive check box is selected. **Warning!** When this check box is selected, Git does not support squashing commits.
Onto	Use this drop-down list to specify the new base for the selected branch. To specify the required commit, type its commit hash or use an expression, for example, of the following structure: `<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>`. Refer to the Git commit naming conventions for details. If no commit is specified, the HEAD of the selected branch is used as the new base.
Validate	Click this button to check that the commit specified in the Onto field exists and view which files were affected in it.
From	Use this drop-down list to specify the commit starting from which you want to apply the branch to the new base. Type the required commit hash or use an expression, for example, of the following structure: `<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>`. Refer to the Git commit naming conventions for details. To apply the entire branch, leave the field empty.
Validate	Click this button to check that the commit specified in the From field exists and view which files were affected in it.
Show Tags	Select this check box to have tagged commits included in the Onto and From drop-down lists.
Show Remote Branches	Select this check box to have branches in the remote repository included in the Onto drop-down list.
Merge Strategy	From this drop-down list, select the merge strategy. The available options are: – Default – Resolve - select this option if you need to resolve two HEADs, one of which is the current branch and the other HEAD is the branch from which you pulled changes. When this option is selected, the 3-way merge algorithm is applied. – Recursive - the default merge strategy for pulling one branch. Select this option if you need to resolve two HEADs by applying the 3-way merge algorithm and there are more than one common ancestor that can be used for 3-way merge. – Octopus - the default merge strategy for pulling more than one branch. Merges that require resolving conflicts manually are not performed. – Ours - select this option if you need to supersede old development history of side branches. By applying this strategy any number of HEADs can be resolved but the result of the merge is always the HEAD of the current branch. – Subtree - a modified recursive strategy.
Do not use merge strategies	When this check box is selected, no merge strategy is applied during rebase.
Rebase	Click this button to initiate rebasing according to the defined settings.

Interactive Rebase Dialog

VCS | Git | Rebase

The dialog box opens when you click Rebase in the Rebase branches dialog box, with the Interactive check box selected.

Use this dialog box to define the order in which commits should be applied, squash or edit commits before applying, skip commits that contain extraneous changes, etc.

Item	Description
Action	Use this drop-down list to define the action that must be applied to the selected commit. The available options are: – Pick: select this option to apply the selected commit as is. – Edit: select this option to edit the files and/or the commit message before applying the selected commit. – Skip: select this option to ignore the selected commit. – Squash: select this option to combine the selected commit with the previous commit. – Reword: select this option if you want to edit the commit message for the selected commit before applying it. – Fixup: select this option to combine the selected commit with the previous one, and construct a commit message from the previous commit message with the "fixup!" prefix.
Commit	This read-only field displays the hash of the selected commit.
Comment	This read-only field shows the comment supplied for the selected commit.
View	Click this button to view the files affected in the selected commit.
Move Up/Move Down	Use these buttons to change the order in which commits should be applied.

Git Reset Dialog

View | Tool Windows | Log - Context menu of a selected commit - Reset Current Branch to Here

Use this dialog box to reset the current branch head to the selected commit, and update the working tree and the index according to the selected mode. The following options are available:

- Soft: The modified files will not be changed, and the differences will be staged for commit.
- Mixed: The modified files will not be changed, and the differences will be unstaged. This mode is used by default and is identical to the `git reset` command.
- Hard: The modified files will be reverted to the selected commit, and all uncommited changes will be lost.
- Keep: The modified files will be reverted to the selected commit, but local changes will be preserved.

Reset Head Dialog

VCS | Git | Reset Head

Use this dialog box to specify the commit you want to reset the current HEAD to and define the reset method.

Item	Description
Git Root	In this drop-down list, select the path to the local repository in which you want to reset the HEAD of a branch.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository. The contents of the field depend on the selection in the Git Root drop-down list.
Reset Type	Use this drop-down list to define the reset method to use. The available options are: – Mixed - the default strategy. When this option is selected, the index is reset while the working tree is not, which means that changed files are preserved but not marked for commit. You are presented with a report of what has not been updated. – Soft - when this option is selected, the index and the working tree are not affected, only the HEAD pointer is moved to the specified commit. Your current state with any changes remains different from the commit you are switching to. All the changes are "staged" for committing. – Hard - when this option is selected, both the working directory and the index are changed to the specified commit.
To Commit	In this text box, specify the commit you want to reset the current HEAD to. Type the desired commit hash or use an expression, for example, of the following structure: `<branch>~<number of commits backwards between the current HEAD and the required commit>` . Refer to the Git commit naming conventions for details.
Validate	Click this button to check that the commit specified in the To Commit field exists and view which files were affected in it.
Reset	Click this button to initiate resetting the HEAD to the specified commit accotding to the defined method.

Stash Dialog

VCS | Git | Stash Changes

Use this dialog box to specify parameters for saving local modifications in a new stash.

Item	Description
Git Root	From this drop-down list, select the path to the local repository in which you want to stash modifications.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository.
Message	In this text box, provide a description of the new stash. This description will be displayed in the Unstash Changes dialog box to help you select the required stash.
Keep Index	Select this check box to have IntelliJ IDEA bring the changes staged in the index to your working tree for examination and testing.
Create Stash	Click this button to have the specified local modification saved in a new stash.

Tag Dialog

VCS | Git | Tag Files

Use this dialog box to assign tags to commits and commit objects.

Item	Description
Git Root	In this drop-down list, select the path to the local repository in which you want to tag a commit or a commit object.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository.
Tag Name	In this text box, type the name of the new tag.
Force	Select this check box to assign an existing tag to another commit or object. This check box is enabled only if you type the name of an existing tag in the Tag Name text box whereupon IntelliJ IDEA displays an error message.
Commit	In this text box, specify the commit or object which you want to tag. Type the desired commit hash or use an expression, for example, of the following structure: `<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>`. Refer to the Git commit naming conventions for details.
Validate	Click this button to check that the commit specified in the Commit field exists and view which files were affected in it.
Message	In the text box, provide a description of the commit to be tagged. Tip Fill in this text box to create an annotated tag.
Create Tag	Click this button to have the specified commit or commit object tagged.

Unstash Changes Dialog

VCS | Git | Unstash Changes

Use this dialog box to specify parameters for applying stashed modifications to your working copy.

Item	Description
Git Root	From this drop-down list, select the path to the local repository in which you want to apply the stashed modifications.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository.
Stashes	From this list, select the stash that you want to apply. Each item is supplied with a number, an indication of the branch in which it was created, and the description provided during the stash creation.
View	Click this button to open the Paths affected in commit dialog box where you can learn which files are affected in the selected stash.
Drop	Click this button to remove the selected stash from the Stashes list.
Clear	Click this button to remove all the stashes from the Stashes list.
Pop Stash	Select this check box to have the selected stash removed from the list after it is applied.
Reinstate Index	Select this check box to have the stashed index modifications applied as well.
As New Branch	If you want to create a new branch on the basis of the selected stash, type the name of the new branch in this list box. When the list box is filled in, the Pop Stash and Reinstate Index check boxes are selected and disabled.
Apply Stash	Click this button to have the specified modifications applied to the working copy.
Pop Stash	Click this button to have the specified modifications applied to the working copy and then removed from the list. The button is available only if the Pop Stash check box is selected.

Update Project Dialog (Git)

VCS | Update Project

In this dialog box, choose the strategy to synchronize your local repository with the remote storage and to choose the way to clean your working tree before update.

Item	Description
Update Type	In this area, choose the method to use for synchronization. The strategy will be applied to all Git version control roots. The available options are: – Merge: choose this option to have the merge strategy applied. The result is identical with that of running `git fetch ; git merge` or `git pull --no-rebase`. – Rebase: choose this option to have the rebase strategy applied. The result is identical with that of running `git fetch ; git rebase` or `git pull --rebase`. – Branch Default: choose this option to have the default command for the branch applied. The default command is specified in the `branch.<name>` section of the `.git/config` configuration file.
Clean working tree before update	In this area, specify the method to save your changes while cleaning your working tree before update. The changes will be restored after the update is completed. The available options are: – Using Stash: choose this option to have the changes saved in a Git stash, so you can apply patches with stashed changed even outside IntelliJ IDEA, because they are generated by Git itself. – Using Shelve: choose this option to have the changes saved on a shelf. Shelving is a IntelliJ IDEA internal operation, patches generated from shelved changes are normally applied (unshelved) inside IntelliJ IDEA. Applying shelved changes outside IntelliJ IDEA is also possible but requires additional steps.
Do not show this dialog in the future	Select this check box to have IntelliJ IDEA update your project silently in the future using the specified Update Type and Clean working copy method. To have IntelliJ IDEA show this dialog box before update again: 1. Open the Version Control - Confirmation page of the Settings dialog box. 2. In the Display Option dialogs when these commands are invoked area, select the Update check box.

Mercurial Reference

In this part:

- Clone Mercurial Repository Dialog
- Create Mercurial Repository Dialog
- Merge Dialog (Mercurial)
- New Bookmark Dialog
- Pull Dialog
- Push Dialog
- Switch Working Directory Dialog
- Tag Dialog (Mercurial)
- Update Project Dialog (Mercurial)

Clone Mercurial Repository Dialog

Checkout from Version Control | Mercurial

Use the dialog box to set up a local repository by downloading the data from a remote repository.

Item	Description
Mercurial Repository URL	In this text box, type the URL of the remote repository which you want to clone.
Test	Click this button to check that connection to the remote repository has been established successfully.
Parent Directory	In this text box, specify the directory where you want IntelliJ IDEA to create a folder for your local Mercurial repository. Type the path manually or click the Browse button ... and choose the desired directory in the dialog that opens .
Directory Name	In this text box, type the name of the new folder into which the repository will be cloned. Warning! The parent directory must not contain a folder with the specified name.
Clone	Click this button to start cloning the specified repository.

Create Mercurial Repository Dialog

VCS | Create Mercurial Repository

Use this dialog box to create a local Mercurial repository in the folder of your choice.

Item	Description
Create repository for the whole project	Select this option to have a repository initialized in the project root directory. This option is helpful if you want to put the entire project under Mercurial control.
Select where to create repository	Select this option to have a repository initialized in one of the folders below the project root. With this option, you can have folders of your project under control of different version control systems. In the text box below, specify the folder to create the repository in. Type the path manually or click the Browse button and choose the desired folder in the dialog that opens .

Merge Dialog (Mercurial)

VCS | Mercurial | Merge

context menu of the Editor - Mercurial | Merge

VCS | Mercurial | Branches - <branch name> - Merge

context menu of the Editor - Mercurial | Branches - <branch name> - Merge

Use this dialog box to merge the current working directory to a named branch , light-weight branch (bookmark) , or a specific changeset identified by a tag, hash, or revision number.

By default, Mercurial requires that before merge the current working directory should be clean, that is, it should not contain any uncommitted changes. Otherwise the merge operation fails and IntelliJ IDEA shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg merge <target branch, bookmark, or changeset> -C` to discard the uncommitted changes.

Item	Description
Repository	From this drop-down list, choose the repository to run the merge in. The contents of the Branch, Tag, and Bookmark drop-down lists are updated to show the branches, tags, and bookmarks that are available in the selected repository.
Merge with	In this area, choose the branch, bookmark, or changeset to merge with. – Branch: choose this option to switch to another line of development identified by a branch name and merge to the branch head . Choose the desired branch from the drop-down list which shows all the named branches available in the current repository. – Tag: choose this option to merge to a changeset to which you have previously assigned a tag identifier . Choose the relevant tag from the drop-down list. The list shows both local tags (from `.hg/localtags`) and global tags (from `.hgtags`). – Bookmark: choose this option to switch to another line of development which is identified by a bookmark and merge to its head. Choose the relevant bookmark from the drop-down list which shows all the available light-weight branches in the current repository. – Revision: choose this option to merge to a specific changeset identified by its hash or revision number. In the text box, type the relevant revision number or paste the hash. To copy a hash, open the Log tab of the Version Control tool window, select the relevant branch and revision, and then choose Copy Hash on the context menu of the selection.

New Bookmark Dialog

VCS | Mercurial | Branches | New Bookmark

context menu of the Editor - Mercurial | Branches | New Bookmark

---

Use this dialog box to establish a new light-weight Mercurial branch (bookmark). The bookmark will immediately appear in the Branches pop-up. You can create both active and inactive bookmarks. By default, IntelliJ IDEA creates an active bookmark, so you are immediately switched to the new bookmark and it is marked with a tick in the Branches pop-up. If you do not want to want to move your development to the new bookmark right now, you can create an inactive bookmark and switch to it later. Note, that tracking and updating is available only for the bookmark that is currently active. You can have only one active bookmark. For details, see http://mercurial.selenic.com/wiki/Bookmarks and Managing Mercurial Branches and Bookmarks .

Item	Description
Bookmark Name	In this text box, type the name of the book,ark. This identifier will always point at the head of the new light-weight branch as you commit changes. You can use this name to identify the head of the relevant light-weight branch when during update or merge.
Inactive	– Clear this check box to activate the new bookmark and thus enable tracking and updating the light-weight branch the bookmarks identifies. The check box is cleared by default. – Select this check box to have an inactive bookmark created, that is, to remain in the current light-weight branch (bookmark) or named branch and switch to the new bookmark later.

Pull Dialog

VCS | Mercurial | Pull Changesets

Use this dialog box to specify parameters for fetching changes from a remote repository and applying them to a local repository.

Item	Description
Pull From	In this text box, specify the URL address of the remote repository to fetch the changes from.

Push Dialog

VCS | Mercurial | Push Changesets

Use this dialog box to specify arguments for uploading changes to a remote repository from the local repository.

Item	Description
Destination Repository URL	In this text box, specify the URL address of the target remote repository.

Options	In this area, specify the changes to be uploaded and configure the push procedure. The available options are:
	– Revision - select this check box to have a specific changeset uploaded. By default, IntelliJ IDEA will upload the latest revision (tip). To have another revision uploaded, type its number in the text box.
	– Branch - select this check to have a branch pushed entirely and choose the relevant branch from the drop-down list.
	– Force push - select this check box:
	  – If you have created a local branch and want to upload it to the remote repository.
	  – If you do not want to update you local repository before push.

Switch Working Directory Dialog

VCS | Mercurial | Update to

Use this dialog box to update the current working directory to a named branch , light-weight branch (bookmark) , or a specific changeset identified by a tag, hash, or revision number.

By default, Mercurial requires that before update the current working directory should be clean, that is, it should not contain any uncommitted changes. Otherwise the update operation fails and IntelliJ IDEA shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg update <target branch, bookmark, or changeset> -C` to discard the uncommitted changes.

Item	Description
Repository	From this drop-down list, choose the repository to run the update in. The contents of the Branch, Tag, and Bookmark drop-down lists are updated to show the branches, tags, and bookmarks that are available in the selected repository.
Switch to	In this area, choose the branch, bookmark, or changeset to switch to. – Branch: choose this option to switch to another line of development identified by a branch name and update to the branch head . Choose the desired branch from the drop-down list which shows all the named branches available in the current repository. – Tag: choose this option to update to a changeset to which you have previously assigned a tag identifier . Choose the relevant tag from the drop-down list. The list shows both local tags (from `.hg/localtags`) and global tags (from `.hgtags`). – Bookmark: choose this option to switch to another line of development which is identified by a bookmark and update to its head. Choose the relevant bookmark from the drop-down list which shows all the available light-weight branches in the current repository. – Revision: choose this option to update to a specific changeset identified by its hash or revision number. In the text box, type the relevant revision number or paste the hash. To copy a hash, open the Log tab of the Version Control tool window, select the relevant branch and revision, and then choose Copy Hash on the context menu of the selection.
Overwrite locally modified files (no backup)	If you are going to update to another branch, bookmark, or changeset and you have any uncommitted changes in the current line of development, technically there can be two ways to treat them. The uncommitted changes can be either committed before update or abandoned (cleaned). By default, Mercurial requires that before update the current working directory should be clean, that is, it should not contain any uncommitted changes. Otherwise the update operation fails and IntelliJ IDEA shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg update <target branch, bookmark, or changeset> -C` to discard the uncommitted changes. Use the Overwrite locally modified files (no backup) check box to prevent failures during update when the current working copy is not clean. – Select the check box to abandon any uncommitted local changes. – Clear the check box if you are sure that the current working directory is clean.

Tag Dialog (Mercurial)

VCS | Mercurial | Tag Repository

context menu of the Editor - Mercurial | Tag Repository

Use this dialog box to create a global tag that identifies the tip of a repository, which is the most recently changed heade in this repository. The created tag will be stored in the file `.hgtags` and tracked by Mercurial.

Item	Description
Select repository to tag	From this drop-down list, choose the repository whose tip you want to tag. The list shows the location of the repositories under the project root.
Tag name	In this text box, type the name of the tag. By this name, you will be able to find the tag in the Log Tab of the Version Control tool window.

Update Project Dialog (Mercurial)

To access this dialog, click VCS in the main menu, and select Update Project from the popup. Alternatively, you can use the `Ctrl+T` shortcut.

In this dialog, select how you want to synchronize your local repository with the central storage.

Option	Description
Pull	Select this option to pull new changesets from the remote repository to the local one. This option can be deselected if the pull operation is performed by other means, for example via a script. The result is identical with that of running the `hg pull` command.
Update Strategy	In this section, select the synchronization method. This strategy will be applied to all Mercurial version control roots. The available options are: – Only Update: select this option to apply the update strategy. The local working directory will be updated to the latest available changeset. The result is identical with that of running the `hg update` command. It is recommended to select this option only if there are no conflicting changes or multiple heads, and if the latest changeset is a descendant or ancestor of the working directory's parent. Otherwise, the update opration will be aborted with errors. – Merge: select this option to apply the merge strategy. The latest changeset from the central repository will be incorporated into the current tip in your working directory. The result is identical with that of running the `hg merge` command. – Commit after merge without conflicts: select this option if you want to commit the resulting changeset after the merge operation has completed successfully. – Rebase: select this option to apply the rebase strategy. Your local changes will be detached, the working directory will be synchronized with the central repository, and then the local changes will be appended on top of the new remote changes. Note To be able to use this method, you need to enable the Rebase extension in the configuration file for your repository (for instructions on how to create configuration files, refer to hgrc).
Do not show this dialog in the future	Select this option to have IntelliJ IDEA update your project silently in the future using the specified update strategy. To invoke this dialog before an update: 1. Open the Settings / Preferences Dialog by choosing File | Settings for Windows and Linux or IntelliJ IDEA | Preferences for OS X, and click Confirmation under Version Control. 2. On the Confirmation page that opens, select the Update checkbox in the Display option dialogs when these commands are invoked area.

Perforce Reference

In this part:

- Edit Jobs Linked to Changelist Dialog
- Integrate File Dialog (Perforce)
- Link Job to Changelist Dialog
- Perforce Options Dialog
- Update Project Dialog (Perforce)

Edit Jobs Linked to Changelist Dialog

VCS | Show Changes View - Local

View | Tool Windows | Changes - Local

VCS | Commit Changes

The dialog box opens in the following cases:

- When you select a changelist and then select Edit Associated Jobs from the context menu.
- When you click the button in the Perforce area of the Commit Changes dialog box.

Use the dialog box to to search for Perforce jobs, link jobs to the selected changelist, and detach currently linked jobs.

Item	Tooltip and Shortcut	Description
	Unlink selected jobs	Click this button to detach the selected job from the changelist.
	Search	Click this button to open the Link Job to Changelist dialog box, where you can search for available jobs, view their details, and link the desired job to the changelist.
	Find and link job matching the pattern	Click this button to start quick search for the job that matches the pattern specified in the text box and attach the job to the changelist. In the text box, specify the exact name of the desired job or a search pattern according to the Perforce jobs syntax rules .

Tip If only one job matching the pattern is found, it is attached to the changelist automatically. Otherwise, to select a job among several available jobs, click and find the desired job using the Link Job to Changelist dialog box.

Close	Click this button to save the specified settings and leave the dialog box.

Integrate File Dialog (Perforce)

VCS | Integrate Project

Context menu of a file or directory | Perforce | Integrate File/Directory

Use this dialog box to integrate changelists from one branch spec to another.

Item	Description
Branch Spec	Select the branch spec that will be used for change integration. Consider the following: – If the Reverse option is enabled, changes are integrated from the selected branch to the local copy. – If the Reverse option is disabled, changes are integrated from the local copy to the selected branch.
Integrate changelist	Use this option to invoke the Changes Browser, where you can select the changelist that will be integrated into the current branch/local copy.
Store Changes To Changelist	Specify the changelist where the integrated changes should be stored.
Revert unchanged files before sync (p4 revert -a)	Select this option to revert unchanged files.
Run resolve automatically after the sync (p4 resolve -am)	Select this option to automatically resolve the files that can be resolved without conflicts.

Link Job to Changelist Dialog

The dialog box opens when you click the button in the Edit Jobs Linked to Changelist dialog box.

Use this dialog box to search for available jobs, view their details, and link jobs to the changelist.

Tip When you need to attach only one job to a changelist, you can use the quick search functionality. This requires that you know the exact name of the job or at least can specify a search pattern for it.

Specify search parameters

Use the controls in this area for specifying various criteria to limit the search output. Follow the Perforce jobs syntax rules . The specified values are joined in the generated command line query via the `AND` operation.

Tip At least one of the fields should be filled in.

Item	Description
Job name pattern	In this text box, type the desired job name search pattern.
Status	Use this drop-down list to specify the status of the job you are looking for. The available options are: – * - when this option is selected, all the jobs that match the remaining search criteria are displayed, regardless of their job statuses. Tip If you specify Status as the only criterion and select this option, all the jobs that are currently present on the Perforce server will be retrieved. – Open – Closed – Suspended
User name pattern	In this text box, specify the search pattern or the exact name of the user who created the desired job.
Date before/Data after	Use these text boxes to specify the time period the desired job is created in. The appropriate formats are `yyyy/mm/dd` or `yyyy/mm/dd:hh:mm:ss`.
Description pattern	In this text box, type the desired job description search pattern.
Search	Click this button to start searching for jobs that match the specified criteria.

Search results

Use this area to view the details of found jobs, select the desired job, and attach it to the changelist.

Item	Description
Search results	The list contains the jobs found according to the specified search criteria. When you select a job, the read-only area shows its details.
OK	Click this button to link the selected job to the changelist.

Perforce Options Dialog

File | Settings | Version Control | Configure | Perforce for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Configure | Perforce for OS X

Ctrl+Alt+S

In this dialog box, configure connection to the specified Perforce server.

Item	Description
Perforce is online	Select this check box to enable establishing connection to the Perforce server. If there is a connection but the server does not respond (for example, because the server is backing up currently), you can disable such attempts by clearing this check box, or IntelliJ IDEA will suggest to do that after a timeout. After that the version control-specific operations will be disabled.
Use P4CONFIG or default connection	Use server information and user credentials from the P4CONFIG environment variable or default Perforce client connection. Otherwise, specify port, user, client, and password.
Charset	Select the charset corresponding to the one set on the server.
Dump Perforce commands to *IDEA_Home\bin\p4.output*	Log Perforce commands in the specified file.
Use login authentication	Toggle authentication.
Try to log in silently	Skip prompt dialog. This option works, when login authentication is required.
Use native Perforce API	Speed up connection to the server using a special library.
Path to P4 executable	Specify the path to the Perforce client executable file.
Test connection	Make sure that connection to the Perforce server is established.
Show branching history	See branches for files in the dialogs showing File History. When you work with several branches, it is recommended to enable this option so that the file branches are correctly displayed.

Update Project Dialog (Perforce)

VCS | Update Project

Ctrl+T

Context menu of a file or directory | Perforce | Update File/Directory

VCS | Perforce | Update File/Directory

Use this dialog box to update the local working copy of a file, directory, or project with a revision from the repository.

Item	Description
Revert unchanged files before sync (p4 revert -a)	Select this check box to discard changes made to open files. This option corresponds to the Perforce `revert` command. See p4 revert reference for details.
Force sync (-f)	Select this check box to forcibly copy files from the depot into the workspace. This option corresponds to the Perforce `sync` command. See p4 sync reference for details.
Run resolve automatically after the sync (p4 resolve -am)	Select this check box to resolve conflicts between file revisions. This option corresponds to the Perforce `resolve` command. See p4 resolve reference for details.
Do not show this dialog in the future	If this check box is selected, the specified actions will be performed silently in future.

Subversion Reference

In this part:

- Authentication Required
- Changes Browser
- Check Out from Subversion Dialog
- Configure Subversion Branches
- Create Branch or Tag Dialog (Subversion)
- Import Into Subversion
- Integrate Project Dialog (Subversion)
- Integrate to Branch
- Lock File Dialog (Subversion)
- Mark Resolved Dialog (Subversion)
- Select Branch
- Select Repository Location Dialog (Subversion)
- Set Property Dialog (Subversion)
- Subversion Options Dialog
- Subversion Working Copies Information Tab
- SVN Repositories
- Update Project Dialog (Subversion)

Authentication Required

VCS | Browse Subversion Repository

Use this dialog to specify your credentials and gain access to the Subversion repository. The dialog box is opened, when you add a new repository location, or attempt to browse a repository.

Item	Description
Authentication realm	This read-only area displays the URL and name of the repository.
User name	By default, this field shows the current user name. You can change the name as required.
Password	Type the password, defined for your Subversion account.
Save credentials	Select this check box to preserve the specified user name and password.

Changes Browser

VCS | Integrate Project

The dialog box opens when you select the Specified options and click the … button.

Use the dialog box to select which revision to use in integration.

Check Out From Subversion Dialog

VCS | Checkout From Version Control | Subversion

Use this dialog box to create local working copies.

In this topic:

– Toolbar

– Main Controls

– Context Menu

Toolbar

Item	Tooltip and Shortcut	Description
	Add Repository Location	Click this button to configure a new repository location .
	Edit Location URL	Click this button to edit the URL address of the selected repository.
	Discard Location	Click this button to discard selected repository location.
	Show/Hide Details	Click this button to display the details for each node below the repository location (changelist number, user name, date and time of the last change).
	Refresh Ctrl+F5	Click this button to refresh the view.
	Collapse All Ctrl+NumPad -	Click this button to have all the nodes below all the repository locations collapsed.

Repositories

In this area, manage the available repositories and select the locations to check out contents from.

Item	Description
Repositories	Use this tree view to explore and manage the available repository locations. If necessary, right-click a node and choose the relevant item from its context menu.
Checkout	Click this button to check out the contents of the selected node to the specified location.

Context Menu

Item	Description
New	Select this menu item to configure a new repository location , or a new remote folder in the selected repository location.
Checkout	Select this menu item to check out the contents of the selected node.
Checkout	Select this menu item to check out the contents of the selected node.
Compare With	Select this menu item to compare the selected node with the specified branch.
Browse Changes	Select this menu item to view changes that match the specified criteria (author, time range, and revision).
Export	Select this menu item to export the contents of the selected repository or folder to the specified destination. The exported contents are not under version control.
Branch or Tag	Select this menu item to create a branch or tag of the selected folder.
Move or Rename	Select this menu item to change name of the selected folder.
Delete	Select this menu item to delete the selected folder from the repository location.
Copy URL Ctrl+Alt+Insert	Select this menu item to put the URL string to the clipboard.
Refresh	Select this menu item to synchronize to the repository.
Edit Location URL	Select this menu item to edit the selected repository location.
Discard Location	Select this menu item to discard the selected repository location.

SVN Checkout Options Dialog

VCS | Checkout From Version Control | Subversion

The dialog box is the third page of the Checkout From Subversion Wizard. In this dialog box, specify the revision to check out and configure the check-out procedure.

Item	Description
Checkout	This read-only field shows the selected source repository.
Destination	From this list, select the directory to create the working copy in. Choose one of the available folders or click the Browse button and select the relevant folder in the dialog box that opens.
Update / Switch to revision	In this area, specify the revision to check out. The available options are: – HEAD - select this option to have the latest revision checked out. – Specified - select this option to have IntelliJ IDEA check out a specific earlier revision. Type the revision number in the text box or click the Browse button and select the relevant revision from the Changes Browse that opens.
Depth	Use this drop-down list to specify the range of recursion into subdirectories. The available options are: – Empty - select this option to involve only the current file. – Files - select this option to involve the files in the folder. – Immediates - select this option to involve direct children of the current file. – Infinity - select this option to enable full recursion.
Include external locations	Select this check box to have externals included in the working copy.

Configure Subversion Branches

Select Branch pop-up dialog box - Configure Branches

Use this dialog box to compose a list of branches you work with.

Item	Description
Trunk location	In this text box, specify the URL address of the trunk in the repository. If necessary, click the Browse button to open the Select Repository Location dialog box and select the required trunk in the repository structure tree.
Branch locations	In this list, select the URL address of the folders in which the required branches are stored.
Add	Click this button to add a branch to the list. The Select Repository Location dialog box opens where you can select the required branch in the repository structure tree.
Remove	Click this button to remove the selected branch from the list.

Create Branch or Tag Dialog (Subversion)

VCS | Subversion | Create Branch or Tag

In this dialog box, set the arguments for creating a branch or a tag on the basis of a local working copy or a repository version.

Item	Description
Copy from	In this section, specify the source folder to create a branch or tag from. The source of the copy can be taken from the local working copy or from the repository.
Working Copy	Click this option to create a branch or tag on the basis of your local working copy. Type the path in the text box or click the Browse button and select the desired directory in the dialog that opens .
Repository Location	Click this option to create a branch or tag on the basis of the repository. Do one of the following: – Type the URL of the repository location in the text box. – Click the browse button and select the source repository location. – Click the button to use the project home directory.
Revision	In this section, specify the source revision to create a branch or tag from. The available options are: – HEAD - select this option to have a branch or tag created on the basis of the HEAD revision. – Specified - select this option to have a branch or tag created on the basis of a specific revision. Type the revision number in the text box manually or click the Browse button and select the desired revision in the Changes Browser dialog box, that opens.
Copy to	Use this section to define the target folder for a branch or tag. The available options are: – Branch or Tag - select this option to have the selected revision copied to a specific branch or tag. In the Base URL text box, specify the base URL of the branch or tag. In the Name text box, specify the name of the new branch. – Any location - select this option to have to have the selected revision copied to a location of your choice. In the text box below, specify the URL of any valid location. Type the URL manually or click the Browse button and specify the desired location in the Select Repository Location dialog box, that opens.
Comment	Type some meaningful description in the text area.

Import into Subversion

VCS | Import into Subversion

Use this dialog box to specify the options for importing data into Subversion.

In this topic:

– Toolbar
– Main Controls
– Context Menu

Toolbar buttons

Item	Tooltip and Shortcut	Description
	Add Repository Location	Click this button to configure a new repository location .
	Edit Location URL	Click this button to edit the URL address of the selected repository.
	Discard Location URL	Click this button to discard the selected repository location and remove it from the list.
	Show/Hide Details	Click this button to display the details for each node below the repository location (changelist number, user name, date and time of the last change).
	Refresh `Ctrl+F5`	Click this button to refresh the view.
	Collapse All `Ctrl+NumPad -`	Click this button to have all the nodes below all the repository locations collapsed.

Main Controls

Item	Description
Repositories	Use this tree view to explore and manage the available repository locations. Right-click nodes and examine context menus.
Import	Click this button and in the dialog that opens , select the directory whose contents should be imported to the selected repository location.

Context Menu

Item	Description
New	Select this menu item to configure a new repository location , or a new remote folder in the selected repository location.
Checkout	Select this menu item to check out the contents of the selected node.
Compare With	Select this menu item to compare the selected node with the specified branch.
Browse Changes	Select this menu item to view changes that match the specified criteria (author, time range, and revision).
Export	Select this menu item to export the contents of the selected repository or folder to the specified destination. The exported contents are not under version control.
Branch or Tag	Select this menu item to create a branch or tag of the selected folder.
Move or Rename	Select this menu item to change name of the selected folder.
Delete	Select this menu item to delete the selected folder from the repository location.
Copy URL `Ctrl+Alt+Insert`	Select this menu item to put the URL string to the clipboard.
Refresh	Select this menu item to synchronize to the repository.
Edit Location URL	Select this menu item to edit the selected repository location.
Discard Location	Select this menu item to discard the selected repository location.

Integrate Project Dialog (Subversion)

VCS | Integrate Project

Use this dialog box to integrate differences between two branches in the Subversion repository into a local working copy.

Item	Description
Source 1	In this text box, specify the URL address of the first branch to compare. If necessary, click the Browse button … and select the desired URL from the Select Repository Location dialog box.
Source 2	In this text box, specify the URL address of the second branch to compare. If necessary, click the Browse button … and select the desired URL from the Select Repository Location dialog box.
Revision	For each source, specify the revision to use. The possible options are: – HEAD - select this option to use the Head revision of the source. The Head revision is suggested by default – Specified - select this option to use a revision different from the Head revision. Specify the required revision in the text field. If necessary, click the … button and select the revision from the Changes Browser dialog box.

Tip Based on the sources and revisions you specify, the difference between source 2 and source 1 is calculated and applied to the local working copy.

Use ancestry	Select this check box to take the ancestry of the Source1 and Source2 URLs into consideration when comparing revisions. If the check box is not selected, only the contents of the files are compared.
Try merge, but make no changes	Select this check box to enable the `--dry-run` switch of the `svn` command. If this check box is not selected, the sources are merged silently.
Depth	Use this drop-down list to specify the range of recursion into subdirectories. The available options are: – Empty - select this option to involve only the current file. – Files - select this option to involve the files in the folder. – Immediates - select this option to involve direct children of the current file. – Infinity - select this option to enable full recursion.

Integrate to Branch

View | Tool Windows | Version Control | Repository Tab | Context menu of a changelist or file |

Subversion | Integrate to Branch

Use this dialog to specify the options for integrating changes into a branch.

ItemTooltip	Description	
Source branch URL		This read-only field shows the URL address of the source branch.
Target branch URL		This read-only field shows the URL address of the target branch.
Integrate into working copy		From this list, select the path to the local working copy into which the changes will be integrated.
Ignore whitespaces		Select this option if whitespaces are not important.
Try merge, but make no changes		Select this option to preview merge results by enabling the `--dry-run` switch of the `svn` command. If this option is unchecked, sources are merged silently.
+	Add	Click this button to add a working copy to the list.
−	Remove	Click this button to remove the selected working copy from the list.

Lock File Dialog (Subversion)

VCS | Subversion | Lock

Context menu of a file | Subversion | Lock

Use this dialog box to lock file when it is necessary to avoid overwriting changes.

Item	Description
Lock Comment	In this text box, describe the reason for locking the file and some additional comments, if necessary.
Steal existing lock	Select this check box to override the lock previously set on the desired file by someone else.
Do not show this dialog in the future	Select this check box to suppress displaying this dialog box and have files and folders locked silently. To have IntelliJ IDEA show this dialog box before locking files or folders again: 1. Open the Version Control - Confirmation page of the Settings dialog box. 2. In the Display Option dialogs when these commands are invoked area, select the Checkout check box.

Mark Resolved Dialog (Subversion)

VCS | Subversion | Mark Resolved

Project tool window | context menu of a file | Subversion | Mark Resolved

Local Changes tab of the Version Control tool window | context menu of a file | Subversion | Mark Resolved

Project tool window | context menu of a file | Subversion | Mark Resolved

Version Control tool window - Merged with conflicts list | context menu of a file | Subversion | Mark Resolved

Use this dialog box to have IntelliJ IDEA consider conflicts in a file or directory resolved. This operation is most often required after merging text or property conflicts manually.

Item	Description
Files and directories	The list shows all the files and directories where merge or updated resulted in conflicts that IntelliJ IDEA cannot resolve automatically. When you have examined the conflicts resolved them manually or considered irrelevant, you need to tell IntelliJ IDEA that these files are no longer conflicting. To appoint a file for marking as free from conflicts, select the check box next to it.
Select All	Click this button to have all the items in the list appointed for marking as resolved.
Deselect All	Click this button to clear the list of candidates for marking as resolved.
Mark Resolved	Click this button to have IntelliJ IDEA treat all the selected items as conflict free. The dialog box closes, whereupon the Local Changes tab of the Version Control tool window shows the affected files as updated and available for submitting to the server.

Select Branch

Version Control tool window | Repository Tab | Merge Info Pane - Browse

Version Control tool window | Subversion Working Copies Information Tab | Merge from

Update Project/Directory dialog box - Browse

The pop-up dialog box opens when you click the Browse button ... or press `Shift+Enter` to select the path to the target branch.

Use this dialog box to select the relevant branch or working copy.

Item	Description
Trunk	Choose this option to set the current trunk as the target branch or working copy.
Branches	Choose this option to select the relevant branch in the Branches list.
Tags	Choose this option to select the relevant tag in the Tags list.
Configure Branches	Choose this option to open the Configure Subversion Branches dialog box and compose a list of branches you work with.

Select Repository Location Dialog (Subversion)

VCS | Subversion | Branch or Tag | Copy From | Repository location

VCS | Subversion | Branch or Tag | Copy To | Any location

Item	Description
Repositories	Use this tree view to explore and manage the available repository locations. Right-click nodes and examine context menus.
Copy as	Specify the name under which the file or folder will be stored in branch.

Set Property Dialog (Subversion)

VCS | Subversion | Set Property

Use this dialog to define SVN-specific properties for the files and folders under SVN version control (ignore list, externals etc.)

Item	Description
Property name	Enter custom property name in the text field, or use the drop-down list to select one of the pre-defined properties.
Set property value	Click this radio-button to set value for the specified property name in the text area. Properties that accept multiple values, such as an ignore list, can be entered on multiple lines.
Delete property	Click this radio-button to remove selected property from the list.
Update properties recursively	Check this option, if you want to apply the property to every file and directory under the selected directory.

Subversion Options Dialog

File | Settings | Version Control | Subversion

Item	Description
Use system default Subversion configuration directory	Store Subversion configuration files in the system default directory: `user_home\Application Data\Subversion`
Subversion configuration directory	Remove the content of the corresponding directory in the Subversion configuration directory. You may need to clear the authorization information from the configuration file, for example, when your credentials have changed.
Clear authentication cache	Remove the content of the corresponding directory in the Subversion configuration directory. You may need to clear the authorization information from the configuration file, for example, when your credentials have changed.

Subversion Working Copies Information Tab

View | Tool Windows | Version Control - Subversion Working Copies Information

Use this tab to configure the format of your working copies.

Tip The tab is only available, when the current project sources are entirely or partially under Subversion control.

The tab displays a list of all detected directories under Subversion control supplied with information on the formats used.

Item	Description
Refresh	Click this button to get the information on all the detected Subversion working copies up-to-date.
Root Path	This read-only field shows the full path to the directory.
URL	This read-only field shows the URL address of the remote directory the selected local copy is mapped to.
Format	This read-only field shows the actual Subversion format used in the selected directory.
Change	Click this link to open the Convert Working Copy Format dialog box, where you can select the desired format option.
Depth	This read-only field shows the range of recursion into subdirectories specified in the Update dialog box.
Working Copy Root	This read-only field is displayed only if the directory in question is the root of a working copy.
Configure Branches	Click this link to open the Configure Subversion Branches dialog box, where you can view and update the list of branches to work with.
Merge from	Click this link to open the Select Branch pop-up dialog box and appoint the source of changes to merge to the current directory.

SVN Repositories

VCS | Browse Subversion Repository

Use this tool window to view, add, and edit the location of SVN repositories.

Toolbar

Item	Tooltip and shortcut	Description
	Add Repository Location	Click this button to configure a new repository location . The New Repository Location dialog box opens, where you can select a repository URL in the drop-down list that contains previously added URL addresses.
	Edit Location Url	Click this button to edit the URL of the selected repository.
	Discard Location	Click this button to remove the selected repository from the list.
	Show/Hide Details	Click this button to display the details for each node under the repository location (changelist number, user name, date and time of the last change).
	Refresh Ctrl+F5	Click this button to refresh the view.
	Expand All/Collapse All Ctrl+NumPad Plus	Click this button to expand/collapse all nodes.
	F1	Click this button to show the corresponding IntelliJ IDEA help page.
	Close	Click this button to close the tool window.

Context Menu

Item	Description
New	Select this menu item to configure a new repository location , or a new remote folder in the selected repository location. The New Repository Location dialog box opens where you can select a repository URL in the drop-down list that contains previously added URL addresses.
Show History	Select this item to open the Version Control tool window with the history of the selected repository location.
Checkout	Select this menu item to check out the contents of the selected node.
Compare with	Select this menu item to compare the selected node with the specified branch.
Browse changes	Select this menu item to view changes that match the specified criteria (author, time range, and revision).
Export	Select this menu item to export the contents of the selected repository or folder to the specified destination. The exported contents are not under version control.
Branch or Tag	Select this menu item to create a branch or tag of the selected folder.
Move or Rename	Select this menu item to change name of the selected folder.
Delete	Select this menu item to delete the selected folder from the repository location.
Copy URL	Select this menu item to put the URL string to the clipboard.
Refresh	Select this menu item to synchronize to the repository.
Edit location Url	Select this menu item to edit the selected repository location.
Discard location	Select this menu item to discard the selected repository location.

Update Project Dialog (Subversion)

VCS | Update Project

`Ctrl+T`

Context menu of a file or directory | Subversion | Update File/Directory

VCS | Subversion | Update File/Directory

Use this dialog box to update the local working copy of a file, directory, or project with a revision from the repository.

Item	Description
Update/Switch to specific Url	– Select this check box to synchronize your local working copy with a specific repository. Specify the source repository either in the URL text box through its full Url address or in the Use Branch text box through the branch name. – Clear this check box to bring the changes from the repository that corresponds to the current working copy.
Use Branch	In this text box, specify the location of the required repository through its branch name. Click the Browse button … to choose the required branch in the Select Branch dialog box that opens. **Note** 1. To use this method of specifying the required repository, you need to configure a list of branches you work with. If you have not done it yet, click Configure Branches in the Select Branch dialog box. 2. The text box is enabled only when the Update/Switch to specific Url check box is selected.
Url	In this text box, specify the location of the required repository through its full URL address. Click the Browse button … to open the Select Repository Location dialog box. The text box is enabled only when the Update/Switch to specific Url check box is selected.
Update/Switch to specific revision	Select this check box to synchronize your local working copy with a specific revision different from the HEAD revision. The Update/Switch to specific revision text box becomes enabled. In this text box, specify the number of the revision to be used. Click the Browse button … to open the Changes Browser dialog box. By default, IntelliJ IDEA suggests to update your local working copy the HEAD revision. This option corresponds to the `--non-recursive (-N)` switch of the Subversion `update` command.
Depth	Use this drop-down list to specify the range of recursion into subdirectories. The available options are: – Working copy - select this option to get files/directories from repository subtrees that have not been checked out yet. – Empty: select this option to involve only the current file. – Files: select this option to involve the files in the folder. – Immediates: select this option to involve direct children of the current file. – Infinity: select this option to enable full recursion.
Force Update	Select this check box to have local files replaced with the files from the repository even if the local files have modifications and thus abandon the local modifications.
Update administrative information only in changed subtrees	This option only applies to working copies older than SVN 1.7 managed by SVNKit. During synchronization with the server (update), SVN locks your working copy one subtree after another by creating empty `lock` files in the corresponding administrative `.svn` directories. After that, SVN starts comparing file hashes to detect which local files need to be synchronized. When this option is selected, SVN first checks if any files from a subtree have been modified on the server, and locks this subtree (i.e. creates a `.svn/lock` file) only if such files are detected. This approach improves performance but may cause concurrency issues, for example, with antiviral software.
Ignore Externals	Select this check box if you do not want IntelliJ IDEA take into account externals definitions during update.
Do not show this dialog in the future	Select this check box to have IntelliJ IDEA perform future updates silently. To have IntelliJ IDEA show this dialog box before update again: 1. Open the Version Control - Confirmation page of the Settings dialog box. 2. In the Display Option dialogs when these commands are invoked area, select the Update check box.

Apply Patch Dialog

VCS | Apply Patch

Use the dialog box to restore changes that were preserved in a patch file in the specified directory.

Item	Tooltip and Shortcut	Description
Patch file name		In this text box, specify the name of the *.patch file to be applied. Type the fully qualified name or click the Browse button and locate the desired patch file using the Select Patch File dialog box, that opens.
	Group by Directory Ctrl+P	Use this button to toggle between the flat view and the directory tree view. Select the check boxes next to the changes that you want to be applied.
	Expand All/Collapse All Ctrl+NumPad Plus Ctrl+NumPad -	Use these buttons to expand/collapse all nodes
	Map base directory	In the dialog that opens , select the directory relative to which file names in the patch file will be interpreted. You can map a base directory to a single file, directory, or to a selection.
	Show Differences Ctrl+D	Click this button to open the Differences viewer for files that shows the differences between your local working copy, the repository version, and the patch. Use the buttons Compare Previous File and Compare Next File to have the files in patch compared in a chain. Tip If the patch cannot be applied without conflicts, the lines with conflicts are highlighted with red.
	Strip Directory	Use this button to apply the changes to files located in different directories from the ones specified in the patch. Clicking this button removes one slash in the path to the target file. Click the button as many times as many leading directories you need to strip. The number of removed slashes is indicated in square brackets.
	Restore Directory	Use this button to revert the last strip directory action. Click the button as many times as many previously stripped leading directories you need to restore.
	Reset Directories	Use this button to revert all strip directory actions in the selection.
	Remove Directories	Click this button to have all the leading directories stripped and have the changes applied to the file with the specified name in the base directory.
	Refresh	Click this button to synchronize the tree with the current state of the file system.
Summary		This section displays summary information for the currently selected changelist (the number of modified, new, and deleted files).
Existing Changelist		Select this option to add the patched files to an existing changelist and select the desired changelist from the drop-down list.
New Changelist		Select this option to create a new changelist and add the patched files to it. – Name: in this text box, type the name of the new changelist. By default, it is the name of the current patch. – Comment: in in this text box, type the comment to the new changelist. – Make this changelist active: select this check box to have IntelliJ IDEA automatically give the active status to the new change list immediately after the changes are restored in it. When this check box is cleared, the current active changelist remains active. See Changelist for details. – Track Context: select this check box to have IntelliJ IDEA preserve the context of the task associated with the new changelist on its deactivation and restore the context when the changelist becomes active. See Managing Tasks and Context for details.

Checkout from TFS Wizard

VCS | Checkout from Version Control | TFS

Use this wizard to download the files from a TFS server according to the settings from a new or an existing workspace .

The menu item and the wizard are available only when the TFS Integration plugin is installed and enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins .

In this part:

– Checkout from TFS Wizard: Checkout Mode
– Checkout from TFS Wizard: Source Server
– Checkout from TFS Wizard: Choose Source and Destination Paths
– Checkout from TFS Wizard: Source Workspace
– Checkout from TFS Wizard: Choose Source Path
– Checkout from TFS Wizard: Summary

Checkout from TFS Wizard: Checkout Mode

VCS | Checkout from Version Control | TFS

On this page, choose the way to map the files and folders on your server with their local copies. These mappings are referred to as workspaces . You can either specify new mappings and have a new workspace generated or use mappings from an existing workspace.

Item	Description
Create workspace automatically	Choose this option to have IntelliJ IDEA generate a workspace for you, with the folder mapping based on your choice of the local and remote paths .
Workspace name	In this text box, type the name of the workspace to be generated.
Choose workspace manually	Choose this option to have the data downloaded according to the mappings from an existing workspace.

Checkout from TFS Wizard: Source Server

VCS | Checkout from Version Control | TFS - Create workspace automatically

On this page, choose the server to download the data from.

Item	Description
Team servers	In this list box, manage the list of the servers you have access to and select the server to download the data from.
Add	Click this button to open the Add Team Foundation Server dialog box, and specify the address of your TFS server and the credentials to connect to it.
Remove	Click this button to delete the selected server from the list.
TFS Proxy	Click this button to open the Set TFS proxy for server... dialog box where you can specify the parameters for accessing the selected server via Proxy.

Checkout from TFS Wizard: Choose Source and Destination Paths

VCS | Checkout from Version Control | TFS - Create workspace automatically

The page opens when you select the server to download the data from on the Source Server page and click Next.

On this page, specify the remote folder to download the data from and map it to a local folder where the data will be downloaded to. The data is downloaded recursively, that is, the structure of subfolders under the selected source node is reproduced locally.

Item	Description
Source Path	In this area, specify the folder on the server to download the data from. Select the folder in the tree, and Source Path read-only field displays the path to it relative to the server root.
Destination path	In this text box, specify the local folder to download the sources to. Type the path to the folder manually or click the Browse button and select the folder in the dialog box, that opens. The data is downloaded recursively, that is, the structure of subfolders under the source node is repeated locally.

Checkout from TFS Wizard: Source Workspace

VCS | Checkout from Version Control | TFS - Choose workspace manually

On this page, specify the workspace to add the imported files and folders to. The data will be downloaded according to the mappings defined in the selected workspace. The data is downloaded recursively, that is, the structure of subfolders under the selected source node is reproduced locally.

You can use an existing workspace as is, or update it as necessary, or even create an entirely new workspace manually.

Item	Description
Server/Workspace	This read-only field shows the URL addresses of TFS servers you have access to and workspaces available on these TFS servers.
Workspace comment	This read-only field shows the descriptions of workspaces on the servers you have access to.
Team Servers	Use the buttons in this area to manage the list of available servers and workspaces and configure access to them. – Add: click this button to open the Add Team Foundation Server dialog box where you can specify the parameters for establishing connection to a TFS server. TFS uses NTLM authentication , so native Windows applications (that is, Microsoft Team Explorer) authenticate silently with system credentials. IntelliJ IDEA users must always specify their username and password because of limitations posed by Java Runtime. – Remove: click this button to remove the selected server from the list. – Reload workspaces: click this button to have the list of available workspaces refreshed. – TFS Proxy: click this button to open the Set TFS proxy for server... dialog box where you can specify the parameters for accessing the selected server via Proxy. – Check-in Policies: click this button to open the Edit Check-in Policies dialog box where you can manage the list of check-in policies to be applied.
Workspaces	Use the buttons in this area manage the list of available workspaces and update the workspaces, when applicable. – Create: click this button to open the Create Workspace dialog box for creating a new workspace. – Edit: click this button to open the Edit Workspace dialog box for editing the selected workspace. – Delete: click this button to remove the selected workspace from the list.

Checkout from TFS Wizard: Choose Source Path

VCS | Checkout from Version Control | TFS - Choose workspace manually

The page opens when you choose an existing workspace on the Source Workspace page and click Next.

On this page, specify the folder on the server to download data from.

The page shows a tree of folders on the server that you have access to. When you select the required folder, $product$ tells you which local folder it is mapped to in the current workspace .

Checkout from TFS Wizard: Summary

VCS | Checkout from Version Control | TFS - Create workspace automatically

VCS | Checkout from Version Control | TFS - Choose workspace manually

This last, read-only, page of the wizard shows the name of the workspace to be used or generated, the URL address of the server, the source folder to download data from, and the local folder to save the downloaded data in. Review the details and click Finish.

Create Patch Dialog

VCS | Create Patch

View | Tool Windows | Version Control - Local Changes - Context menu of a file or changelist - Create Patch

---

Use this dialog box to generate a patch file for the specified changelist or files.

Use this dialog box to create a patch from the selected changelist or files.

This dialog box consists of several areas:

- Modified files pane
  - Toolbar
- Commit Message pane
  - Toolbar
- Details pane
  - Toolbar

Modified files pane

This section contains a list of files that were modified since the last commit. All files in this list are selected by default. Deselect the check-boxes next to the files that you want to exclude from the patch.

Toolbar

Item	Tooltip and Shortcut	Description	Available in
	Show Differences Ctrl+D	Click this button to open the Differences dialog box that points at the inconsistencies between your local working copy of the selected file and the file in the repository.	All VCSs
	Refresh Changes Ctrl+F5	Click this button to reload the Changed files tree view so it is up-to-date.	All VCSs
	Show Unversioned Files	Click this button if you want to see newly added files that have not been added to version control yet under the Unversioned Files node.	ALl VCSs
	Add to VCS Ctrl+Alt+A	Click this button to move the files selected under the Unversioned Files node to the active changelist, so that they are added to your version control system during the commit.	All VCSs
	Move to Another Changelist F6	Click this button to add the selected file(s) to another changelist. The Move to Another Changelist dialog box opens where you can select an existing changelist or create a new one.	All VCSs
	Delete	Click this button to delete the selected file.	
	Ignore	Click this button to leave the selected files unversioned.	All VCSs
	Rollback	Click this button to revert all changes made to the local working copy of the selected file(s).	All VCSs
	Jump to source F4	Click this button to open the source code of the selected file in the editor.	All VCSs
	Revert Unchanged Files	Click this button to revert the files that have not been modified locally.	Subversion Perforce
	Group by Directory Ctrl+P	Click this button to toggle between the flat view and the directory tree view.	All VCSs
 	Expand or collapse all nodes Ctrl+NumPad Plus Ctrl+NumPad -	Click these buttons to expand or collapse all nodes in the directory tree. These buttons are not available in flat view.	All VCSs
Change list	N/A	From this drop-down list, select the changelist that contains the modified files to be checked in or included in the patch. The active changelist is selected by default.	All VCSs

The summary under the modified files pane shows statistics on the currently selected changelist, such as the number of modified, new and deleted files. This area also shows how many files of each type are shown, and how many of them will be included in the patch.

Commit Message pane

The comment you enter in this area will be used as the name of the patch file.

Toolbar

Icon and Tooltip	Shortcut	Description
Check commit message spelling	N/A	Click this icon if you want to automatically check the spelling in your commit messages.

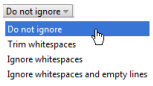
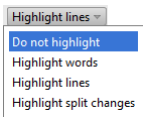
Commit Message History	Ctrl+M	Click this icon to invoke the Commit Message History dialog that contains a list of your twenty-five last commits and the corresponding commit messages.

Details pane

The Details pane is hidden by default. To unfold it, click the arrow button next to the pane title.

In this pane you can explore the differences between the base repository version of the selected file, and the version you want to include in the patch.

Toolbar

Item	Tooltip and Shortcut	Description
	Previous Difference / Next Difference Shift+F7 F7	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, IntelliJ IDEA suggests to click the arrow buttons F7 / Shift+F7 once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings . This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.
	Compare Previous/Next File Alt+Left Alt+Right	Click these buttons to compare the local copy of the previous/next file with its update from the server. Tip These buttons are only available when there is more than one file in the selected changelist.
	Jump to Source F4	Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.
Viewer type		Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to – Edit, for example, type text. Note that one can change text only in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line. – Perform the Apply/Append/Revert actions.
Whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text. – Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default. – Trim whitespaces: ("\t", " ") , if they appear in the end and in the beginning of a line. – If two lines differ in trailing whitespaces only, these lines are considered equal. – If two lines are different, such trailing whitespaces are not highlighted in the By word mode. – Ignore whitespaces: white spaces are not important, regardless of their location in the source code. – Ignore whitespaces and empty lines: the following entities are ignored: – all whitespaces (as in the 'Ignore whitespaces' option) – all added or removed lines consisting of whitespaces only – all changes consisting of splitting or joining lines without changes to non-whitespace parts. For example, changing a b c to a \n b c is not highlighted in this mode.
Highlighting mode		Select the way differences granularity is highlighted. The available options are: – Highlight words: the modified words are highlighted – Highlight lines: the modified lines are highlighted – Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. For example, A \n B vs. A X \n B X will be treated as two changes instead of one. – Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	Collapse unchanged fragments	Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.
	Synchronize scrolling	Click this button to scroll both differences panes simultaneously. If this button is released, each pane can be scrolled independently.
	Disable editing	Click this button to enable editing of the local copy of the selected file, which is disabled by default. When editing is enabled, you can make last-minute changes to the modified file before committing it.
	Editor settings	Click this button to open a drop-down list of available options. Select or clear these options to show or hide line numbers, indentation guides, white spaces, and soft wraps.
	Show diff in external tool	Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff Tools settings page.
	Help F1	Click this button to show the corresponding help page.

Note that the options listed above are available for text files only. IntelliJ IDEA cannot compare binary files, so most commands will be unavailable for them.

After you've selected the files you want to commit, click the Create Patch button and specify the patch file options in the dialog that opens.

Commit Changes Dialog

VCS | Commit Changes

View | Tool Windows | Version Control - Local Changes - Context menu of a file or change list - Commit Changes

---

Use this dialog box to commit (check in) changes from the selected changelist to the repository and, optionally, to create a patch file.

This dialog box consists of several areas:

- Modified files pane
  - Toolbar
- Commit Message pane
  - Toolbar
- VCS-specific controls
- Before Submit / Before Commit section
- After Submit / After Commit section
- Details pane
  - Toolbar
- Submit / Commit button

The options available in this dialog depend on the version control system you are using.

Modified files pane

This section contains a list of files that were modified since the last commit. All files in this list are selected by default. Deselect the check-boxes next to the files that you want to exclude from the current commit.

Toolbar

Item	Tooltip and Shortcut	Description	Available in
	Show Differences `Ctrl+D`	Click this button to open the Differences dialog box that points at the inconsistencies between your local working copy of the selected file and the file in the repository.	All VCSs
	Refresh Changes `Ctrl+F5`	Click this button to reload the Changed files tree view so it is up-to-date.	All VCSs
	Show Unversioned Files	Click this button if you want to see newly added files that have not been added to version control yet under the Unversioned Files node.	ALl VCSs
	Add to VCS `Ctrl+Alt+A`	Click this button to move the files selected under the Unversioned Files node to the active changelist, so that they are added to your version control system during the commit.	All VCSs
	Move to Another Changelist `F6`	Click this button to add the selected file(s) to another changelist. The Move to Another Changelist dialog box opens where you can select an existing changelist or create a new one.	All VCSs
	Delete	Click this button to delete the selected file.	
	Ignore	Click this button to leave the selected files unversioned.	All VCSs
	Rollback	Click this button to revert all changes made to the local working copy of the selected file(s).	All VCSs
	Jump to source `F4`	Click this button to open the source code of the selected file in the editor.	All VCSs
	Revert Unchanged Files	Click this button to revert the files that have not been modified locally.	Subversion Perforce
	Group by Directory `Ctrl+P`	Click this button to toggle between the flat view and the directory tree view.	All VCSs

	Expand or collapse all nodes
	`Ctrl+NumPad Plus`
	`Ctrl+NumPad -`
		Click these buttons to expand or collapse all nodes in the directory tree. These buttons are not available in flat view.	All VCSs

Change list	N/A	From this drop-down list, select the changelist that contains the modified files to be checked in or included in the patch. The active changelist is selected by default.	All VCSs

The summary under the modified files pane shows statistics on the currently selected changelist, such as the number of modified, new and deleted files. This area also shows how many files of each type are shown, and how many of them will be committed.

Commit Message pane

In this area, enter a comment to the current commit. You cannot commit your changes until you enter some description in the Commit Message field.

This comment will also be used as the name of the patch file, if you decide to create a patch.

Toolbar

Icon and Tooltip	Shortcut	Description
Check commit message spelling	N/A	Click this icon if you want to automatically check the spelling in your commit messages.
Commit Message History	Ctrl+M	Click this icon to invoke the Commit Message History dialog that contains a list of your twenty-five last commits and the corresponding commit messages.

VCS-specific controls

The controls in this section are located in the top-right part of the dialog, and contain the options that are specific for the version control system you are using.

Item	Description	Available for
Author	Use this drop-down list to select the author of the changes that you are going to commit. This may be useful when you are committing changes made by another person.	Git
Amend commit	Select this check box to replace the previous commit with the current changes (see Git Basics: Undoing Things for details).	Git, Mercurial
Keep files locked	Select this check box to keep the changed files locked after they are checked in.	Subversion
Jobs	These controls are available only if you select the Enable Perforce Jobs Support check box on the Perforce settings page . Use the controls in this area to search for Perforce jobs , link jobs to the selected changelist, and detach the currently linked jobs. – Unlink selected jobs: click this button to detach the selected job from the changelist. – Edit associated jobs: click this button to open the Edit Jobs Linked to Changelist dialog where you can search for available jobs, view their details, and link jobs to the selected changelist. – Find and link job matching the pattern: click this button to start quick search for the job that matches the pattern specified in the text box and attach the job to the changelist. In the text box, specify the exact name of the job or a search pattern according to the Perforce jobs syntax rules . Tip If only one job matching the pattern is found, it is attached to the changelist automatically. Otherwise, to select a job among several available jobs, click the button and find the desired job using the Edit Jobs Linked to Changelist dialog box. The list box in the bottom of the area displays the jobs that are currently attached to the selected changelist.	Perforce

Before Submit / Before Commit section

Use the controls in this area to define which additional actions you want IntelliJ IDEA to perform before committing the selected files.

These controls are available for the following version control systems:

– Git
– CVS
– Subversion
– Perforce

Item	Description
Reformat code	Select this check box to perform code formatting according to the Project Code Style settings .
Rearrange code	Select this check box to rearrange your code according to the arrangement rules preferences .
Optimize imports	Select this check box to remove redundant import statements .
Perform code analysis	Select this check box to run code inspection on the files you are about to commit.
Check TODO (<filter name>)	Select this check box to review the TODO items matching the specified filter. Click the Configure link to choose an existing TODO filter , or open the TODO settings page and define a new filter to be applied.
Cleanup	Select this check box if you want to automatically apply the current inspection profile to the files you are going to commit.
Update copyright	Select this check box to add or update a copyright notice according to the selected copyright profile - scope combination .
Revert unchanged files	Select this check box to revert the files that have not been modified. This option is only available for Perforce.

After Submit / After Commit section

Use the controls in this area to define which additional actions you want IntelliJ IDEA to perform after committing the selected files.

Item	Description	Available for
Run tool	From this drop-down list, select the external tool that you want IntelliJ IDEA to launch after the selected changes have been committed. You can select a tool from the list, or click the Browse button ... and configure an external tool in the External Tools dialog box that opens.	All VCSs
Upload files to	From this drop-down list, select the server access configuration to use for uploading the committed files to a local or remote host, a mounted disk, or a directory. To suppress uploading, choose None. To add a server configuration to the list, click ... and fill in the required fields in the Add Server dialog that opens.	All VCSs

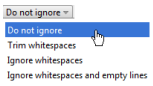
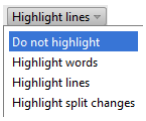
Always use selected server	Select this check box to always upload files to the selected server access configuration. The drop-down list and the check box are only available if the Remote Hosts Access plugin is enabled.	All VCSs
Tag committed files	Select this check box to assign a tag to the committed files and type the name of the tag. To replace a previously assigned tag with a new one, select the Override existing tags option.	CVS
Auto-update after commit	Select this check box to automatically update your project after the commit. Enabling this option will help prevent your working copy against the mixed-revision state . The mixed-revision state of a working copy may affect the Move and Rename refactoring applied to folders, in which case items in revisions different from the moved subtree root will be tracked separately, which can be confusing. When the Auto-update after commit option is enabled: – Merge will fail with an error if the merge target is a mixed-revision working copy. – Your own changes will never cause a 409 conflict.	Subversion

Details pane

The Details pane is hidden by default. To unfold it, click the arrow button next to the pane title.

In this pane you can explore the differences between the base repository version of the selected file, and the version you are about to commit.

Toolbar

Item	Tooltip and Shortcut	Description
	Previous Difference / Next Difference Shift+F7 F7	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, IntelliJ IDEA suggests to click the arrow buttons F7 / Shift+F7 once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings . This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.
	Compare Previous/Next File Alt+Left Alt+Right	Click these buttons to compare the local copy of the previous/next file with its update from the server. Tip These buttons are only available when there is more than one file in the selected changelist.
	Jump to Source F4	Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.
Viewer type		Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to – Edit, for example, type text. Note that one can change text only in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line. – Perform the Apply/Append/Revert actions.
Whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text. – Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default. – Trim whitespaces: `("\t", " ")` , if they appear in the end and in the beginning of a line. – If two lines differ in trailing whitespaces only, these lines are considered equal. – If two lines are different, such trailing whitespaces are not highlighted in the By word mode. – Ignore whitespaces: white spaces are not important, regardless of their location in the source code. – Ignore whitespaces and empty lines: the following entities are ignored: – all whitespaces (as in the 'Ignore whitespaces' option) – all added or removed lines consisting of whitespaces only – all changes consisting of splitting or joining lines without changes to non-whitespace parts. For example, changing `a b c` to `a \n b c` is not highlighted in this mode.
Highlighting mode		Select the way differences granularity is highlighted. The available options are: – Highlight words: the modified words are highlighted – Highlight lines: the modified lines are highlighted – Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one. – Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	Collapse unchanged fragments	Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.
	Synchronize scrolling	Click this button to scroll both differences panes simultaneously. If this button is released, each pane can be scrolled independently.
	Disable editing	Click this button to enable editing of the local copy of the selected file, which is disabled by default. When editing is enabled, you can make last-minute changes to the modified file before committing it.
	Editor settings	Click this button to open a drop-down list of available options. Select or clear these options to show or hide line numbers, indentation guides, white spaces, and soft wraps.
	Show diff in external tool	Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff

		Tools settings page.
?	Help	Click this button to show the corresponding help page. F1

Note that the options listed above are available for text files only. IntelliJ IDEA cannot compare binary files, so most commands will be unavailable for them.

Submit / Commit button

Click this button to commit the selected files, or hover your mouse over this button to display one of the following available commit options:

- Commit and Push: select this option to push the changes to the remote repository immediately after the commit. This option is available if you are using Git or Mercurial as a version control system.
- Create MQ Patch: select this option to create an MQ patch based on your changes. This option is only available if you are using Mercurial as a version control system.
- Create Patch: select this option if you want IntelliJ IDEA to generate a patch based on the changes you are about to commit. In the Create Patch dialog that opens, type the name of the patch file and specify whether you need a reverse patch.
- Remote Run: select this option to run your personal build . This option is only available when you are logged in to TeamCity . Refer to TeamCity plugin documentation for details.

Configure Ignored Files Dialog

Version Control tool window | Local Changes -

Use this page to create a list of files and directories that you do not want to put under version control. These can be file names associated with VCS administration, backup files, and any other artifacts that you want to remain unversioned. You can also specify patterns that define files to ignore.

Item	Keyboard shortcut	Description
+	Alt+Insert	Use this icon or shortcut to add an item to the list. The Ignore Unversioned Files dialog box opens where you can type an exact path to a file or directory to be ignored or specify a pattern that defines the names of files and directories to be ignored.
	Enter	Use this icon or shortcut to edit the selected path or pattern in the Ignore Unversioned Files dialog box.
−	Alt+Delete	Use this icon or shortcut to remove the selected path or pattern from the list.

Enable Version Control Integration Dialog

VCS | Enable Version Control Integration

In this dialog box, choose one of the registered Version Control Systems to use in your project and assign it to the Project Root.

The dialog box and the menu item are available only if the project does not use any Version Control System.

Item	Description
Select a version control system to associate with the project root	From this drop-down list, select one of the supported version control systems that you want to associate with the project root.

File Status Highlights

In IntelliJ IDEA each file has its own status marked with a specific color. The file status denotes correspondence of the actual file content with the one marked as 'current'.

In the editor, each line in a file is checked whether it corresponds to the state at the 'current' point and marked with a specific color at the left gutter area.

You can customize the default colors:

- For files - in the File Status page of the Colors and Fonts settings.
- For the lines in the editor - in the General page of the Colors and Fonts settings.

In this section:

- File Status in Views .
- Line Status in the Editor .

File Status in Views

Color	File Status	Description
Black	Up to date	File is unchanged.
Gray	Deleted	File is scheduled for deletion from the repository.
Blue	Modified	File has changed since the last synchronization.
Green	Added	File is scheduled for addition to the repository.
Violet	Merged	File is merged by your VCS as a result of an update.
Brown	Unknown	File exists locally, but is not in the repository, and is not scheduled for adding.
Olive	Ignored	File will be ignored in any VCS operation.
Light brown	Hijacked	File is modified without checkout . This status is valid for the files under Perforce, ClearCase and VSS.
Red	Merged with conflicts	During the last update, file was merged with conflicts.
Lilac	Externally deleted	File is deleted locally, but was not scheduled for deletion, and still exists in the CVS repository.
Dark cyan	Switched	The file is taken from a different branch than the whole project. This status is valid for CVS and SVN.

Line Status in the Editor

Color	File Status	Description
	Modified	Denotes the lines modified since the last synchronization.
	Added	Denotes the lines added since the last synchronization.
	Deleted	Denotes the lines removed since the last synchronization.

New Changelist Dialog

Version Control tool window - +

`Alt+Insert`

Use this dialog box to create a new changelist.

Item	Description
Name	Type the name of the new changelist.
Comment	Type optional comment. When the new changelist will be submitted to the repository, this comment will appear in the Comment text area of the Commit Changes dialog box.
Make this changelist active	Select this check box to have IntelliJ IDEA automatically give the active status to the new change list immediately after the changes are restored in it. When this check box is cleared, the current active changelist remains active. See Changelist for details.
Track context	Select this check box to have IntelliJ IDEA preserve the context of the task associated with the new changelist on its deactivation and restore the context when the changelist becomes active. See Managing Tasks and Context for details.

Patch File Settings Dialog

VCS | Create Patch - Create Patch

View | Tool Windows | Version Control - Local Changes - Context menu of a file or changelist - Create Patch - Create Patch

Use this dialog to configure the patch file settings.

Item	Description
Patch file	Specify the name of the patch file. By default, the text in the Commit Message section of the Create Patch dialog is used as the file name. If the Commit Message section is empty, the default name is `unnamed.patch`.
Reverse patch	Select this option if you want to create a patch that reverts the changes you have made.
Include entire base revision content into patch file	Select this option if you want to store the entire base revision text of the modified file inside the patch file. This option is applicable to non-binary files smaller than 500kB.
Encoding	Select the encoding for the patch file from the drop-down list.

Revert Changes Dialog

VCS | Show Changes View - Local - Context menu of a file or change list - Revert Changes

View | Tool Windows | Changes - Local - Context menu of a file or change list - Revert Changes

---

Use this dialog box to roll back changes that have not yet been committed to the repository.

Toolbar

Item	Tooltip and Shortcut	Description
	Show Differences Ctrl+D	Click this button to open the Differences dialog box that points at the inconsistencies between your local working copy of the selected file and the file in the repository.
	Move to Another Changelist F6	Click this button to add the selected file(s) to another changelist. The Choose Changelist dialog box opens where you can select an existing changelist or create a new one.
	Group by Directory Ctrl+P	Click this button to toggle between the flat view and the directory tree view.
	Expand or collapse all nodes Ctrl+NumPad Plus Ctrl+NumPad -	Click these buttons to expand or collapse all nodes in the directory tree. These buttons are not available in flat view.
	Select All Ctrl+A	Click this button to select all the files in the list or directory tree.

Controls

Item	Description
Changed files	This tree view displays the list of changed files. Select check boxes next to the files to be reverted.
Change list	Use the drop-down list to select the change list that contains the modified files to be reverted. By default, the active change list is suggested.
Delete local copies of added files	Use this check box to revert added files as well as your changes in modified ones.

Select Target Changelist Dialog

VCS | Show Changes View - Repository / Incoming - Context menu of a file or change list - Revert Changes

View | Tool Windows | Changes - Repository / Incoming - Context menu of a file or change list - Revert Changes

Use this dialog box to roll back changes from a certain change list.

Item	Description
Existing Changelist	Choose this option restore the shelved changes in one of the existing changelists. Choose the target changelist from the drop-down list.
New Changelist	Choose this option to create a new changelist and restore the changes from the shelf in it. – Name: in this text box, type the name of the changelist to be created. – Comment: in this text box, type an optional description of the new changelist. – Make this changelist active: select this check box to have IntelliJ IDEA automatically give the active status to the new change list immediately after the changes are restored in it. When this check box is cleared, the current active changelist remains active. See Changelist for details. – Track context: select this check box to have IntelliJ IDEA preserve the context of the task associated with the new changelist on its deactivation and restore the context when the changelist becomes active. See Managing Tasks and Context for details.
Remove successfully applied files from the shelf	– Clear this check box to have IntelliJ IDEA still display already unshelved changes in the Shelf tab so that you can apply them once more if necessary. – When this check box is selected, the changes are not displayed in the Shelf tab after they are unshelved.

Shelve Changes Dialog

VCS | Shelve Changes

Use this dialog box to put changes in the specified files or changelists to the shelf.

Toolbar

Icon	Tooltip and Shortcut	Description
	Show differences Ctrl+D	Click this button to have differences shown.
	Move to another changelist F6	Click this button to open the Choose Changelist dialog box where you can specify the changelist to move the selected file to.
	Group by Directory Ctrl+P	Use this button to toggle between the flat view and the directory tree view.
	Ctrl+NumPad Plus	Click this button to have all nodes expanded.
	Ctrl+NumPad -	Click this button to have all nodes collapsed.
	Rollback	Click this button to roll the changes back.
	Revert Unchanged Files	Click this button to have unchanged files reverted.
	F4	Click this button to open the source code of the selected file in the editor.
	Message History Ctrl+M	Click this button to reuse one of the previous commit messages (comments) from the list.

Controls

Item	Description
Changed files	This tree view displays the list of changed files. Check the files to be included in the patch.
Comment	Type the comment in the text area. This comment is used as the name of the new patch file.
Changelist	Use the drop-down list to select the changelist that contains the modified files to be included in the patch. By default, the active changelist is suggested.
Summary	This section displays summary information for the currently selected changelist (the number modified, new and deleted files).
Before commit	Use this section to define the following options:
Optimize imports	Check this option to remove redundant import statements.
Reformat code	Check this option to perform code formatting according to the Project Code Style settings .
Perform code analysis for affected files	Check this option to run code inspection on the affected files.
Shelve changes	Click this button to perform shelving and close the dialog.

Unshelve Changes Dialog

Version Control Tool window | context menu | Unshelve Changes

Use this dialog box to restore shelved changes from a shelf to a changelist.

Item	Description
Existing Changelist	Choose this option restore the shelved changes in one of the existing changelists. Choose the target changelist from the drop-down list.
New Changelist	Choose this option to create a new changelist and restore the changes from the shelf in it. – Name: in this text box, type the name of the changelist to be created. – Comment: in this text box, type an optional description of the new changelist. – Make this changelist active: select this check box to have IntelliJ IDEA automatically give the active status to the new change list immediately after the changes are restored in it. When this check box is cleared, the current active changelist remains active. See Changelist for details. – Track context: select this check box to have IntelliJ IDEA preserve the context of the task associated with the new changelist on its deactivation and restore the context when the changelist becomes active. See Managing Tasks and Context for details.
Remove successfully applied files from the shelf	– Clear this check box to have IntelliJ IDEA still display already unshelved changes in the Shelf tab so that you can apply them once more if necessary. – When this check box is selected, the changes are not displayed in the Shelf tab after they are unshelved.

Visual SourceSafe Options Dialog

File | Settings | Version Control | Configure | SourceSafe for Windows and Linux

IntelliJ IDEA | Preferences | Version Control | Configure | SourceSafe for OS X

Ctrl+Alt+S

Item	Description
Path to VSS client (ss.exe)	Specify executable file of the Visual SourceSafe client
Path to VSS configuration file	Specify configuration file `srcsafe.ini`. The configuration file `srcsafe.ini` can be located either locally (usually, when a local repository is used) or under a shared network folder. In the latter case, you should use the UNC names.
User name	Enter your VSS user name
Enter user named	Enter your VSS password
Working directories	View the mapping list between VSS project and local working directories
Add	Create a new VSS project and assign a working directory for it in the Add Mapping dialog. In the VSS Project field, specify the directory location within the VSS repository, starting with $/. In the Working directory field, specify the local working folder.
Edit	Change name or working directory of an existing VSS project in the Edit Mapping dialog.
Remove	Delete the selected project from the list.

Java EE Reference

In this part:

- Apply EJB 3.0 Style
- Change EJB Classes Dialog
- Choose Servlet Class
- Create CMP Field
- Create / Edit Relationship
- Edit File Set
- EJB Editor
- EJB ER diagram
- EJB Module Editor
- Generate GWT Compile Report Dialog
- Generate Persistence Mapping - Import Dialogs
- New Bean Dialogs
- New Servlet Dialog
- New Filter Dialog
- New Listener Dialog
- Rename Entity Bean
- Select Accessor Fields to Include in Transfer Object
- Web Services Reference
- XML-Java Binding Reference

Apply EJB 3.0 Style

EJB Tool Window | context menu of a module with an EJB facet | Apply EJB 3.0 Style

Use this dialog box to bring the beans created according to the EJB 1.x or EJB 2.0 specifications into compliance with the EJB 3.0 specification.

Item	Description
EnterpriseBeans to apply EJB 3.0 style to	This table shows the list of Objects encountered in the selected module with EJB facet. Use the check boxes in the left column of the table to include or exclude beans from refactoring. If a certain bean has a home interface, use the check box in the Retain Home interfaces column to convert home interfaces, rather than delete them.
Environment Access	Select the preferred mode of environment access: – Prefer resource injection: click this radio-button to force convert all environment access to resource injection. – Prefer lookup: click this radio-button to force convert all environment access to context lookup. – Leave as is if possible: click this radio-button to leave environment access as is in case it does not interfere with the converted code. If not, IntelliJ IDEA automatically selects the most appropriate method.
Replace JNDI lookup with EJB context lookup	If this check box is selected, the system resources will be retrieved using the `EJBContext.lookup()` . Otherwise, JNDI environment lookup will be used.
Inline injected fields	If this check box is selected, all usages of a context lookup will be replaced with a reference to an an injected field.
Copy metadata from XML descriptor	Select this check box to copy meta information to the `ejb-jar.xml` file.
Delete copied XML tags	Clear this check box to preserve duplicates.
Replace Entity beans with CMP to Persistence Unit	Select this check box, if you want to try to convert outdated entity beans to Container Managed Persistence. **Warning!** Select this option with extreme care because it can render your project incompilable.
Refactor	Click this button to perform refactoring, and close the dialog box.
Preview	Click this button to open tentative refactoring results in the dedicated tab of the Find Tool Window , and close the dialog box.

Change EJB Classes Dialog

EJB Tool Window | context menu of a bean | Jump to Source | Change EJB Classes

Use the dialog box to change properties of existing beans.

Tip The contents of the dialog box are bean type-specific. See comments in the Available in column.

Item	Description	Available in
<ejb-name>	This read-only field shows the basic name used for generating the names of the EJB constituent classes. The rules for generating EJB names are configured in the Java EE Names dialog box.	All bean types
Package	In this text box, specify the fully qualified path to the package where the bean resides or click the Browse button ... and select the desired package in the module tree.	All bean types
EJB Class	This read-only field shows the name of the bean implementation class.	All bean types
Message Listener	In this text box, specify the message listener interface. Type a fully qualified name or click the Browse button ... and select the message listener interface from the list of available interfaces.	Message beans
Primary key class	In this text box, specify the class that will be used to access the primary key of the data source the entity bean is associated with.	Entity beans
CMP version	This read-only field shows the used CMP version. For BMP Entity beans, the field is read-only.	Entity beans
Remote Interface	Select this check box if you want to configure a remote client view of the bean. Tip Selecting this check box enables the Home and Remote text boxes.	Entity beans
Home	In this text box, specify the implementation class for a remote home interface. If necessary, click the Browse button ... to open the Choose EJB Home Interface dialog box, where you can search for the desired interface by name or select it in the project tree.	Entity beans
Remote	In this text box, specify the implementation class for a remote interface. If necessary, click the Browse button ... to open the Choose EJB Remote Interface dialog box, where you can search for the desired interface by name or select it in the project tree.	Entity beans
Local Interface	Select this check box if you want to configure a local client view of the bean. Tip Selecting this check box enables the Local Home and Local text boxes.	Entity beans
Local Home	In this text box, specify the implementation class for a local home interface. If necessary, click the Browse button ... to open the Choose EJB Local Home Interface dialog box, where you can search for the desired interface by name or select it in the project tree.	Entity beans
Local	In this text box, specify the implementation class for a local interface. If necessary, click the Browse button ... to open the Choose EJB Local Interface dialog box, where you can search for the desired interface by name or select it in the project tree.	Entity beans

Choose Servlet Class

Web Module Editor - General - Servlet Initialization Params

The dialog opens when you select a servlet in the Servlets Configured pane and click Change Class in the Servlet Initialization Params that opens.

Use this dialog to select the class that implements the selected servlet.

The dialog contains two tabs:

- Search by Name
- Project

Search by Name Tab

Use the tab to search a relevant class to implement the servlet. Specify the class name or part of the name.

Item	Description
Search pattern area	The text field for typing a part of the name of the relevant class.
Search results area	Shows a list of classes that meet the search pattern. The contents of the area change dynamically as you type.
Include non-project classes	Involves classes outside the current project into the search.

Project Tab

Use this tab to select the relevant class in the project tree.

Create CMP Field

EJB Tool Window | context menu of an entity bean | Jump to Source

This dialog box opens when you click the Add button + in the Entity Bean Specifics section.

Use the dialog box to create a CMP field for the current entity bean.

Item	Description
Name	In this text box, specify the name of the CMP field.
Description	In this text box, provide a description of the CMP field.
	Click this button to open the Description dialog box and type s description of the CMP field there.
Type	From this drop-down list, select the required type for the CMP field. Note Actually, you select the class that implements the type.
	Click this button to open the Choose Class dialog box, where you can select a class that is not available in the Type drop-down list.
Primary Key	Select this check box to set the current CMP field as a primary key.
Generate getter in	Select the relevant check box to specify where the `getter` method of the entity bean will be generated. The available options are: – Local Interface – Remote Interface Note If the bean does not have a local or remote interface, the corresponding check box is disabled and the remaining option is selected.
Generate setter in	Select the relevant check box to specify where the `setter` method of the entity bean will be generated. The available options are: – Local Interface – Remote Interface Note If the bean does not have a local or remote interface, the corresponding check box is disabled and the remaining option is selected.

Create / Edit Relationship

This dialog opens when you draw or double-click a link between entities on an entity-relationship diagram for a persistence unit or a session factory.

Specify the settings for the relationship.

The left-hand and the right-hand parts are for the two sides of the relationship. For a unidirectional relationship, specify the settings only in one of the parts.

Item	Description
Attribute	The name of the field to be added to the corresponding entity and also the name of the relationship.
Multiplicity	The multiplicity on the corresponding side of the relationship.
Optional	Whether the relationship is optional.
Owner	The owner of the relationship.
Fetch Type	See FetchType .
Cascade Type	See CascadeType .

Edit File Set

File | Project Structure - Modules - module - Struts 2 facet

Ctrl+Alt+S, 1

The dialog opens when you click + (Alt+Insert) or (Enter) on the File Sets tab of the Struts 2 Facet page .

Use this dialog to configure a validation file set by selecting the relevant files in the module tree.

Item	Description
File Set name	Use this field to edit the file set name.
Locate	Use this button to add the files that are external to your module. (The Select Path dialog will open.)

EJB Editor

EJB Tool Window | context menu of a bean | Jump to Source

The bean editor is accessible from the EJB tool window only. The editor consists of two tabs: General tab and Assembly Descriptor tab.

The General tab consists of a common part, and a bean specific part. The Assembly Descriptor tab is similar for all bean types.

In this part:

- EJB Editor - General Tab - Entity Bean
- EJB Editor - General Tab - Message Bean
- EJB Editor - General Tab - Session Bean
- EJB Editor - Assembly Descriptor
- Assembly Descriptor Dialogs
- EJB Editor General Tab - Common

EJB Editor - General Tab - Entity Bean

EJB Tool Window | context menu of an entity bean | Jump to Source

This section describes the fields that are specific for the entity beans.

Item	Shortcut	Description
Abstract Schema Name		In this text box, specify the bean reference name that can be used to call the bean from QL queries. This field is available for the CMP Entity Beans only.
Primary Key Class		In this field, specify the class that will be used to access primary key of a datasource the bean is associated with. This field is available for the CMP and BMP Entity Beans.
Reenterant		Select this check box if you need the bean to handle multiple simultaneous, interleaved, or nested invocations that do not interfere with each other. This field is available for the CMP and BMP Entity Beans.
CMP fields		These fields are available for CMP Entity Beans only.
	Insert	Click this button to create a new CMP field .
	F4	Click this button to change CMP field .
	Delete	Click this button to delete selected field with its accessor methods and tags in the deployment descriptor.
	F1	Click this button to show reference page.

EJB Editor - General Tab - Message Bean

EJB Tool Window | context menu of a message-driven bean | Jump to Source

This section describes the fields that are specific for Message beans.

Item	Shortcut	Description
Transaction Type		In this field, specify the object by which the transactions should be managed. The available options are: – Bean – Bean container
Destination Type		In this field, specify the message queue class to which the bean should listen.
Activation Config		Use this section to define the bean behavior.
+	Insert	Click this button to add activation configuration elements.
–	Delete	Click this button to delete selected activation configuration element.
?	F1	Click this button to show reference page.

EJB Editor - General Tab - Session Bean

EJB Tool Window | context menu of a session bean | Jump to Source

This section describes the fields that are specific for Session beans.

Item	Description
Session Type	Select session type (stateful or stateless) depending on the conversational state the bean should have.
Transaction Type	Specify an object the transactions should be managed with. The possible options are a bean and its container.

EJB Editor - Assembly Descriptor

EJB Tool Window | context menu of a bean | Jump to Source

Use this tab to define how the beans are deployed and configured at the target application server.

Item	Shortcut	Description
+	Insert	Click this button to create an element of the deployment descriptor.
	F4	Click this button to change the selected element.
−	Delete	Click this button to delete the selected element.

Configuration setting	Parameters
EJB Environment Entry	– Name – Value – Type – Description
EJB Local Reference	– Name – EJB Home interface – Link – Type – Remote interface – Description
EJB Reference	– Name – EJB Home interface – Link – Type – Remote interface – Description
Web Service Reference	– Name – Type – Service interface – JAX-RPC mapping file – WSDL name – Service QName – Description
EJB Resource Reference	– Name – Type – Auntication – Shareable – Description
EJB Resource Environment Reference	– Name – Type – Description
Message Destination Reference	– Name – Type – Message destination usage – Message destination line – Description

Assembly Descriptor Dialogs

The dialog boxes that open are specific for the various types of assembly descriptors.

EJB Editor General Tab - Common

EJB Tool Window | context menu of a bean | Jump to Source

The fields in the General section are available for all bean types.

Item	Description
Display name	Use this field to change the display name of a bean.
Description	Use this field to type optional textual description of a bean.
EJB Classes	These fields are available for all bean types.
Change EJB Classes	Click this button to open the bean-specific Change Bean dialog , where you can define the classes that comprise a bean: bean class, local and remote interfaces, and primary key class.
Rename EJB and Classes	Click this button to change the name of the bean and propagate this change to all the constituent classes of the bean. This renaming is performed as the Rename Refactoring .

EJB ER diagram

An EJB ER diagram opens on a separate editor tab when you select ER Diagram in the context menu for a module in the EJB tool window .

Use this diagram to view entity beans defined in your module, and also to create and change relationships between them.

– Toolbar

– Context menu

Toolbar

Item	Description
	Click this button to show grid in the diagram background.
	Click this button to align elements against the grid.
	Click this button to increase the scale of the diagram.
	Click this button to decrease the scale of the diagram.
	Click this button to restore the actual size of the diagram.
	Click this button to change the scale to make the contents fit into the current diagram size.
	Click this button to apply the current layout, selected on the context menu of the diagram.
	Click this button to save the diagram in an image file with the specified name and path. The possible formats are: `jpeg` , `png` and `gif` .
	Click this button to print the diagram.
	Click this button to open the diagram preview in a separate frame, where you can configure the page layout, scale, and headings information.

Context menu

Item	Description
New	Use this node to add new EJBs to a module with the EJB facet and configure relationships between CMP Entity Beans.
Layout	Choose this command to select the desired layout from the list.
Show edge labels	Choose this command to show multiplicities of the relationship links in diagram.

EJB Relationship Properties dialog

The EJB Relationship Properties dialog opens when your draw a line between entity beans on an EJB ER diagram . (In this way you create a relationship between the corresponding beans.)

Use this dialog to define the relationship properties, and also to add the corresponding elements to the deployment descriptor.

Item	Description
Relationship Name	Type the name of a new relationship link.
Description	Type optional description. If the text is long, click the ellipsis button, and type the desired text in the editor dialog box.
Role 1,2	Use these sections of the dialog box to specify attributes of each side of a relationship link.
EJB	Select the source and target EJBs from the drop-down lists.
Role name	Type the names of the roles for each side of the relationship link.
Multiplicity	Select multiplicity from the drop-down list.
CMR Field	Select this check box, if you want to create a container-managed relation field `<cmr-field >` in the deployment descriptor. If this option is checked, specify following attributes.
Field name	Type the name of the new CMR field. The `<cmr-field-name>` element is generated in the deployment descriptor.
Field type	Select the type of the new CMR field. The `<cmr-field-type>` element is generated in the deployment descriptor.
Getter / Setter	If the check boxes are selected, the getter and setter methods are generated in the bean implementation class.

EJB Module Editor

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source

---

Use the tabs in this editor to configure the beans contained in a module with an EJB facet, method permissions , and transaction attributes .

If the EJB module editor is invoked from the deployment descriptor, the text editor tab is also available.

EJB Module Editor - General

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source

Use this tab to configure the beans contained in a module with EJB facet.

Item	Description
New	Click this button to add a new bean to a module with EJB facet. The bean type is selected from the submenu. Refer to the section Creating EJB for the detailed description of procedure.
Remove	Click this button to delete the selected bean from the module.
Edit	Click this button to invoke the EJB editor for the selected bean. Refer to the section Editing EJB for the detailed description of procedure.
Help	Click this button to show reference page.

EJB Module Editor - Method Permissions

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source

Use this tab to configure security roles and method permissions for a bean.

Toolbar

Item	Shortcut	Description
	`Insert`	Click this button to define a custom security role by specifying the security role name and optional description in the Add Security Role dialog box. A column for a new security role is added to the table.
	`F4`	Click this button to change the selected security role by choosing the desired role from the submenu, and changing the security role name and optional description in the Edit Security Role dialog box.
	`Delete`	Click this button to delete the selected security role.

Table

Item	description
Name	This column shows beans, their interfaces, and a list of methods that they implement.
Excluded	Select the check boxes in this column for the methods, which you want to add to the list of excluded methods. The selected methods will be added to the `<exclude-list >` section of the deployment descriptor.
Unchecked	Select the check boxes in this column for the methods with unchecked permissions. The selected methods will be marked as `<unchecked/>` in the deployment descriptor.
`<security role >`	Select the check boxes for the methods to assign the selected security roles.

EJB Module Editor - Transaction Attributes

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source

Use this tab to manage the transaction attributes that are used by the EJB container to control the transaction scope when the enterprise bean methods are invoked.

Item	Description
Name	This column shows beans, their interfaces and the list of methods that they implement.
Transaction Attribute	This column displays the transaction attribute for a method, interface or bean. The possible values are: – Mandatory – Never – NotSupported – Required – RequiresNew – Supports

Generate GWT Compile Report Dialog

Tools | Generate GWT Compile Report

Use this dialog box to view the compilation output of a GWT module.

Item	Description
GWT Module	From this drop-down list, select the module to view the output of.
Generate	Click this button to have a new report generated.
View Report	Click this button to have the previous generated report opened in the browser. Tip IntelliJ IDEA informs you how long ago a report was generated last.

Generate Persistence Mapping - Import dialogs

From the Persistence tool window : Right-click a module, persistence unit or session factory, point to Generate Persistence Mapping and select the necessary option.

Specify the settings for generating entity classes and object/relational mappings for them. Depending on the option that you selected, one of the following is used as a source:

- A database schema represented by a data source .
- An EJB facet, if exists in the current project, or, to be more exact, the deployment descriptor file ejb-jar.xml associated with that facet. (Only the <entity> elements are processed.)
- Only for JPA: a Hibernate object/relational mapping file (.hbm.xml). The file should be within the current project.

- General Settings
- Database Schema Mapping
- Generation Settings

General Settings

Item	Description
Choose Data Source	When importing a database schema: Specify the data source to be used as a source of import. ... opens the Data Sources and Drivers dialog which lets you create a new data source.
Choose EJB Facet	When importing an EJB facet: Specify the EJB facet to be used as a source. ... opens the dialog which lets you select the facet. (The EJB facets available in the current project are suggested.)
Choose Hibernate XML	When importing a Hibernate object/relational mapping file: Specify the `.hbm.xml` file to be used as a source. ... opens the dialog which lets you select the file. (The corresponding file must be available in the current project.)
Package	The destination package for your entity classes. ... opens the dialog which lets you select an existing package, or create a new one ( ).
Entity prefix	One or more characters to be used as a prefix for your entity class names.
Entity suffix	One or more characters to be used as a suffix for your entity class names.
Prefer primitive types	When importing a database schema: Prefer primitive field types to object types (e.g. when `int` and `java.lang.Integer` are the alternatives).
Show default relationships	When importing a database schema: If this check box is selected, IntelliJ IDEA analyzes the foreign keys in the tables and suggests to create corresponding relationships (e.g. one-to-one, one-to-many).

Database Schema Mapping

This section is available only when importing a database schema.

Select the tables and columns to be mapped. Edit the names of the entity classes and their persistent fields (the Map As column). Adjust the field types (the Mapped Type column). Use the toolbar for working with relationships and performing other tasks.

Item	Description
+	Create a new relationship between the entities. (The Add Relationship dialog will open.)
	Edit the selected relationship. (The Edit Relationship dialog will open.)
−	Delete the selected relationship.
	Refresh the database schema.
	Select all the tables, fields and relationships.
	Deselect all the tables, fields and relationships.
	Expand all the nodes in the table.
	Collapse all the nodes in the table.

Generation Settings

Item	Description
Add to Persistence Unit / Add to Session Factory	The persistence unit or session factory with which the generated entity classes will be associated. If the Generate Persistence Mapping command was called on a persistence unit or session factory, the info about the entity classes will be added to the current persistence unit or session factory (hibernate.cfg.xml). If the command was called on a module, you can select a target persistence unit or session factory. This may be an existing persistence unit or session factory (hibernate.cfg.xml), or a new one. To create a new persistence unit or session factory, click +. In the case of the persistence unit, just specify its name (a new `<persistence-unit name=""/>` element will be added to `persistence.xml` .) In the case of the session factory, a new Hibernate configuration file will be created. So, in the dialog that opens, select the destination folder, click and specify the file name. `.xml` in the file name may be omitted.
Generate Column Properties	Select the check box for the column properties (e.g. `length` , `nullable`) to be included.
Generate Single Mapping XML	Store object/relational mappings for all the generated entity classes in one XML file. To define the target file: 1. Click +. 2. If suggested, select the target file format (Hibernate mapping or JPA mapping descriptor (JPA ORM)). 3. In the dialog that opens, select an existing mapping file or create a new one. To create a new file, select the destination directory, click and specify the file name. `.xml` in the file name may be omitted.

Generate Separate XML per Entity	Create an individual object/relational mapping file for all the generated entity classes. The file names will be `<ClassName>.xml` for JPA and `<ClassName>.hbm.xml` for Hibernate.
Generate JPA Annotations (Java 5)	Add the mapping information to the source code of the generated entity classes as annotations.

Add / Edit Relationship

From the Import Database Schema dialog : + or in the Database Schema Mapping section.

Use this dialog to specify the settings for a relationship.

Item	Description
Source / Target	Within the relationship, one of the entities (the left-hand one) is referred to as a source and the other one as a target.
Table	The corresponding database table.
Attribute name	The name of the field to be added to the corresponding entity and also the name of the relationship.
Type	The Java type for the corresponding field.
Map Key Column	Select the desired column name from the drop-down list.
Join Table	The join table if used.
Join Columns	Joined table columns for the relationship. Use + and – to manage the list.

New Bean Dialogs

EJB Tool Window | context menu of a module with an EJB facet | New - bean type

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source | New - bean type

---

Use these dialog boxes to create EJBs of various types.

The contents of the dialog boxes are bean type-specific.

In this section:

- New Session Bean Dialog
- New Message Bean Dialog
- New BMP Entity Bean Dialog
- New CMP Entity Bean Dialog

New Session Bean Dialog

EJB Tool Window | context menu of a module with an EJB facet | New - Session Bean

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source | New - Session Bean

---

Use this dialog box to create Session Beans .

Item	Description
<ejb-name>	In this text box, type the basic name to be used for generating the names of the EJB constituent classes. The rules for generating EJB names are configured in the Java EE Names dialog box.
Package	In this text box, specify the fully qualified path to the package where the bean resides. If necessary, click the Browse button and select the desired package in the project tree.
EJB Class	In this text box, specify the name of the bean implementation class. If necessary, click the Browse button to open the Choose EJB Class dialog box, where you can search for the desired class by name or select it in the project tree.

New Message Bean Dialog

EJB Tool Window | context menu of a module with an EJB facet | New - Message Bean

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source | New - Message Bean

---

Use this dialog box to create Message Beans .

Item	Description
<ejb-name>	In this text box, type the basic name to be used for generating the names of the EJB constituent classes. The rules for generating EJB names are configured in the Java EE Names dialog box.
Package	In this text box, specify a fully qualified path to the package where the bean resides. If necessary, click the Browse button ... and select the desired package in the project tree.
EJB Class	In this text box, specify the name of the bean implementation class. If necessary, click the Browse button ... to open the Choose EJB Class dialog box, where you can search for the desired class by name or select it in the project tree.
Message Listener	In this text box, specify the message listener interface. Type a fully qualified name or click the Browse button ... to open the Choose EJB Message Listener Interface dialog box, where you can search for the desired message listener interface by name or select it in the project tree.

New BMP Entity Bean Dialog

EJB Tool Window | context menu of a module with an EJB facet | New - BMP Entity Bean

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source | New - BMP Entity Bean

---

Use this dialog box to create Bean Managed Persistence (BMP) Entity Beans .

Item	Description
<ejb-name>	In this text box, type the basic name to be used for generating the names of the EJB constituent classes. The rules for generating EJB names are configured in the Java EE Names dialog box.
Package	In this text box, specify a fully qualified path to the package where the bean resides. If necessary, click the Browse button ... and select the desired package in the project tree.
EJB Class	In this text box, specify the name of the bean implementation class. If necessary, click the Browse button ... to open the Choose EJB Class dialog box, where you can search for the desired class by name or select it in the project tree.
Primary Key Class	In this text box, specify the class to be be used for accessing the primary key of the data source the entity bean is associated with. If necessary, click the Browse button ... to open the Choose EJB Primary Key Class dialog box, where you can search for the desired class by name or select it in the project tree.
CMP version	This read-only field shows the Container Managed Persistence (CMP) version used.
Remote Interface	Select this check box if you want to configure a remote client view of the bean. Tip Selecting this check box enables the Home and Remote text boxes.
Home	In this text box, specify the implementation class for a remote home interface. If necessary, click the Browse button ... to open the Choose EJB Home Interface dialog box, where you can search for the desired interface by name or select it in the project tree.
Remote	In this text box, specify the implementation class for a remote interface. If necessary, click the Browse button ... to open the Choose EJB Remote Interface dialog box, where you can search for the desired interface by name or select it in the project tree.
Local Interface	Select this check box if you want to configure a local client view of the bean. Tip Selecting this check box enables the Local Home and Local text boxes.
Local Home	In this text box, specify the implementation class for a local home interface. If necessary, click the Browse button ... to open the Choose EJB Local Home Interface dialog box, where you can search for the desired interface by name or select it in the project tree.
Local	In this text box, specify the implementation class for a local interface. If necessary, click the Browse button ... to open the Choose EJB Local Interface dialog box, where you can search for the desired interface by name or select it in the project tree.

New CMP Entity Bean Dialog

EJB Tool Window | context menu of a module with an EJB facet | New - CMP Entity Bean

EJB Tool Window | context menu of a module with an EJB facet | Jump to Source | New - BMP Entity Bean

---

Use this dialog box to create Container Managed Persistence (CMP) Entity Beans .

Item	Description
<ejb-name>	In this text box, type the basic name to be used for generating the names of the EJB constituent classes. The rules for generating EJB names are configured in the Java EE Names dialog box.
Package	In this text box, specify a fully qualified path to the package where the bean resides. If necessary, click the Browse button … and select the desired package in the project tree.
EJB Class	In this text box, specify the name of the bean implementation class. If necessary, click the Browse button … to open the Choose EJB Class dialog box, where you can search for the desired class by name or select it in the project tree.
Primary Key Class	In this text box, specify the class to be used for accessing the primary key of the data source the entity bean is associated with. If necessary, click the Browse button … to open the Choose EJB Primary Key Class dialog box, where you can search for the desired class by name or select it in the project tree.
CMP version	From this drop-down list, select the Container Managed Persistence (CMP) version to be used.
Remote Interface	Select this check box if you want to configure a remote client view of the bean. Tip Selecting this check box enables the Home and Remote text boxes.
Home	In this text box, specify the implementation class for a remote home interface. If necessary, click the Browse button … to open the Choose EJB Home Interface dialog box, where you can search for the desired interface by name or select it in the project tree.
Remote	In this text box, specify the implementation class for a remote interface. If necessary, click the Browse button … to open the Choose EJB Remote Interface dialog box, where you can search for the desired interface by name or select it in the project tree.
Local Interface	Select this check box if you want to configure a local client view of the bean. Tip Selecting this check box enables the Local Home and Local text boxes.
Local Home	In this text box, specify the implementation class for a local home interface. If necessary, click the Browse button … to open the Choose EJB Local Home Interface dialog box, where you can search for the desired interface by name or select it in the project tree.
Local	In this text box, specify the implementation class for a local interface. If necessary, click the Browse button … to open the Choose EJB Local Interface dialog box, where you can search for the desired interface by name or select it in the project tree.

New Servlet Dialog

File | New | Servlet

Specify the settings for the new servlet.

Item	Description
Name	The "root part" of the servlet name. This name (along with its prefix and suffix) will be used either in the `<servlet-name>` element in `web.xml` or in the servlet class `@WebServlet` annotation. You can specify the servlet name prefix and suffix (the <servlet-name> tag Prefix and Suffix fields on the Java EE Names tab of the Editor | Code Style | Java page in the Settings dialog). In that case, the prefix and the suffix are added before and after the specified name automatically.
Package	The name of the package in which the new servlet class should be created. Click ..., Then select an existing package or click to create a new package.
Class	The fully qualified name of the servlet class. The default class name is generated using the package name, the class name prefix, the servlet name, and the class name suffix. The class name prefix and suffix are set in the Settings dialog. (The Servlet Class Prefix and Suffix fields on the Java EE Names tab of the Editor | Code Style | Java page.) If you want to specify an existing class, click ... and select the class in the dialog that opens.
Create Java EE 6 annotated class	If the check box is not selected, the servlet name - class mapping is added to `web.xml`. If the check box is selected, the new servlet class will be `@WebServlet`-annotated and no changes will be made to `web.xml`. If there is no `web.xml` in your project, the check box is selected by default, and you cannot deselect it.

Choose Servlet Package

This dialog opens when you click next to the Package field in the New Servlet dialog .

Select an existing package or click to create a new package. The new servlet class will be created in the specified package.

New Filter Dialog

File | New | Filter

Specify the settings for the new filter.

Item	Description
Name	The "root part" of the filter name. This name (along with its prefix and suffix) will be used either in the `<filter-name>` element in `web.xml` or in the filter class `@WebFilter` annotation. You can specify the filter name prefix and suffix (the <filter-name> tag Prefix and Suffix fields on the Java EE Names tab of the Editor | Code Style | Java page in the Settings dialog). In that case, the prefix and the suffix are added before and after the specified name automatically.
Package	The name of the package in which the new filter class should be created. Click ..., Then select an existing package or click to create a new package.
Class	The fully qualified name of the filter class. The default class name is generated using the package name, the class name prefix, the filter name, and the class name suffix. The class name prefix and suffix are set in the Settings dialog. (The Filter Class Prefix and Suffix fields on the Java EE Names tab of the Editor | Code Style | Java page.) If you want to specify an existing class, click ... and select the class in the dialog that opens.
Create Java EE 6 annotated class	If the check box is not selected, the filter name - class mapping is added to `web.xml` . If the check box is selected, the new filter class will be `@WebFilter` -annotated and no changes will be made to `web.xml` . If there is no `web.xml` in your project, the check box is selected by default, and you cannot deselect it.

New Listener Dialog

File | New | Listener

Specify the settings for the new listener.

Item	Description
Name	The listener name. Used only to generate the default listener class name.
Package	The name of the package in which the new listener class should be created. Click ... Then select an existing package or click to create a new package.
Class	The fully qualified name of the listener class. The default class name is generated using the package name, the class name prefix, the listener name, and the class name suffix. The class name prefix and suffix are set in the Settings dialog. (The Listener Class Prefix and Suffix fields on the Java EE Names tab of the Editor | Code Style | Java page.) If you want to specify an existing class, click ... and select the class in the dialog that opens.
Create Java EE 6 annotated class	If the check box is not selected, a `<listener-class>` element for the listener class is added to `web.xml`. If the check box is selected, the new listener class will be `@WebListener`-annotated and no changes will be made to `web.xml`. If there is no `web.xml` in your project, the check box is selected by default, and you cannot deselect it.

Rename Entity Bean

EJB Tool Window | context menu of an entity bean | Jump to Source

This dialog box opens when you click Rename EJB and Classes in the EJB Classes section.

Use the dialog box to rename an entity EJB and/or the classes that implement it.

Item	Description
<ejb-name>	Type the basic name that will be used to generate the names of the entity EJB constituent classes. The rules for generating EJB names are configured in the Java EE Names settings tab.
Abstract Schema Name	Type the name of the abstract scheme that CMP will use to map to the physical database.
Refactor	Click this button to update the EJB deployment descriptor `ejb-jar.xml` according to the changes made in the fields above. The Rename Entity Beans Classes dialog box opens where you can specify which of the EJB constituent classes must be renamed.
Preview	Click this button to open the Refactoring Preview window with all the occurrences of the EJB constituent classes.

Select Accessor Fields to Include in Transfer Object

EJB Tool Window | context menu of an entity bean | New | Transfer Object

Use this dialog box to select methods to be included in a transfer object for an entity bean.

ItemTooltipDescription		
		Use this button to have methods sorted in the ascending or in the descending order.
	Show Classes	If a bean refers to ancestor classes, clicking this button reveals these classes and their members. Otherwise, the button is not available. This functionality depends on the EJB specification.
	Expand All	Click this button to have all nodes expanded.
	Collapse All	Click this button to have all nodes collapsed.
List of members in the entity bean in question		From the list, select the desired methods. Use the `Ctrl` and `Shift` keys for multiple selection.
Transfer object class name		In this field, specify a name for a transfer object class that matches a certain pattern. Accept the name suggested by IntelliJ IDEA or change it as required.
Generate getter/setter in <class name> interface		Select these check boxes to have accessor methods for the selected fields generated. You can accept the suggested names of the accessor methods or change them as required.
Copy JavaDoc		Select this check box to have a JavaDoc comment to the generated transfer object included, if any.

Web Services Reference

In this section:

- Enable Web Services Support Dialog
- Expose Class as Web Service Dialog
- Generate WSDL from Java Dialog
- Generate Java Code from WSDL or WADL Dialog
- Monitor SOAP Messages Dialog
- Show Deployed Web Services Dialog

Enable Web Services Support Dialog

Tools | Web Services | Enable Web Service Support

In this dialog box, enable support of a particular implementation to develop the server or the client side of a Web service. The dialog box allows for more flexible configuring of Web service development support by choosing a specific platform version to be used. You can also specify whether you need to have a sample class created or not.

Item	Description
Choose Module	From this drop-down list, select the module to enable Web service or Web service client development in. The drop-down list contains only modules with Web development support enabled.
Type	From this drop-down list, select the Web service side to enable support of. The avaialble options are: − Server − Client
Web Service Platform	From this drop-down list, select the Web service platform to use in development.
Add Sample Code	Select this check box to have a sample class generated.
Status	View the information in this read-only field to track and improve discrepancies when configuring the code generation procedure.

Expose Class As Web Service Dialog

`Alt+Enter` - Expose Class as Web Service

The dialog box is available only through a dedicated intention action. To invoke it, position the cursor at the class name and press `Alt+Enter` or click the yellow bulb icon.

Use the dialog box to configure Web service WSDL generation for an entire class, with all its methods exposed as Web service operations and deployed. The contents of the dialog box depend on the Web service type.

Item	Description	Web Service Type
Service Name	Use this drop-down list to specify the name of the service to be published (e.g. empty text for `ROOT` context or `mycontext` for `/mycontext`).	Apache Axis
Service Class Name	In this text box, specify the class to expose as a Web service.	All
Service Namespace	In this text box, type the Web service namespace prefix.	Apache Axis
Service Style	Use this drop-down list to specify the style of the WSDL document to be generated. The available options are: – RPC - select this option to have an rpc WSDL generated. This option is selected by default. – Document - select this option to have a Document WSDL generated. – Wrapped - select this option to have a WSDL generated using the wrapped approach. If you select this option, Literal is automatically pre-selected in the Use Items in Bindings drop-down list.	Apache Axis
Use of Items	From this drop-down list, select how the generated WSDL document should be used. The available options are: – Literal - when this option is selected, the representation of the XML for the request is defined by the XML Schema. – Encoded - select this option to have the SOAP encoding specified in the generated WSDL.	Apache Axis
Target Module		All
Status	View the information in this read-only field to track and improve discrepancies.	All

Generate WSDL from Java Dialog

Tools | Web Services | Generate WSDL From Java Code

Use the dialog box to configure Web service WSDL generation and select the methods to be exposed as Web service operations and deployed. The contents of the dialog box depend on the Web service type.

Item	Description	Web Service Type
Class to Generate WSDL For	This read-only field shows the name of the class	All
Web Service URL	In this text box, specify the URL address the Web service will be available at.	All
Methods for Operation	In this area, specify which methods of the selected class you want to be deployed as Web service operations. – Method to Expose - this column shows a list of all the methods within the selected class. – Add to Deployment - select the check box next to the methods you want to be deployed as operations. If a method cannot be selected, a tooltip explains the reason.	Apache Axis
Web Service Namespace	In this text box, type the Web service namespace prefix.	Apache Axis
SOAP Action	From this drop-down list, select the value for the `soapAction` attribute of the `<wsdlsoap:operation />` field. The available options are: – Default - when this option is selected, the `soapAction` is set according to the operation's meta data (usually to ""). – Operation - when this option is selected, the `soapAction` is set to the operation's name. – None - when this option is selected, the `soapAction` is set to "".	Apache Axis
Binding Style	Use this drop-down list to specify the style of the WSDL document to be generated. The available options are: – RPC - select this option to have an rpc WSDL generated. This option is selected by default. – Document - select this option to have a Document WSDL generated. – Wrapped - select this option to have a WSDL generated using the wrapped approach. If you select this option, Literal is automatically pre-selected in the Use Items in Bindings drop-down list.	Apache Axis
Use Items in Bindings	From this drop-down list, select how the generated WSDL document should be used. The available options are: – Literal - when this option is selected, the representation of the XML for the request is defined by the XML Schema. – Encoded - select this option to have the SOAP encoding specified in the generated WSDL.	Apache Axis
Type Mapping Version	Use this drop-down list to specify the default type mapping registry for mapping the Java class to an XML qualified name, using a specified Serializer. The available options are: – 1.1 indicates SOAP 1.1 JAX-RPC compliant. – 1.2 indicates SOAP 1.1 encoded.	Apache Axis
Status	This read-only field shows whether the specified URL address is correct.	All

Generate Java Code from WSDL or WADL Dialog

The dialog box opens after you create a Java module and enable Web services client development in it. To access the dialog box at any time during the development, select the desired client module in the Project view and choose WebServices | Generate Java Code from Wsdl or Wadl on the context menu.

Use the dialog box to have the client-side XML-Java bindings generated based on the desired WSDL descriptor of the target Web service.

Technically, IntelliJ IDEA generates Java code from WSDL using third party libraries that are controlled through a command line. This command line is assembled of the data you enter in the fields of this dialog box.

Item	Description	Web Service Client Type
Web service wsdl url	Use this drop-down list to specify the location of the target Web service WSDL descriptor.	All
User Name and Password	In these text boxes, type the credentials for accessing the WSDL URL address. The fields are mandatory if the WSDL location requires authentication.	JAX-WS
Output Path	Use this drop-down list to specify the module source directory to place the generated files in.	All
Package Prefix	Use this drop-down list to specify the package for the compiled Java classes.	All
Output Mode	Use this drop-down list to specify whether you want to generate Java code only for the client side or for the server side as well.	Apache Axis
Type Mapping Version	Use this drop-down list to specify the default type mapping registry for mapping an XML qualified name to a Java class, using a specified Deserializer. The available options are: – 1.1 – 1.2	Apache Axis
Allow Extensions	Select this check box to have Java code generated for the extension points contained in the WSDL file.	All
Generate TestCase	Select this check box to have an additional JUnit test case class generated for testing purposes.	Apache Axis
Generate Classes for Schema Arrays	Do one of the following: – Select this check box to have classes generated for schema arrays. – Clear the check box to have Java arrays used.	Apache Axis
Generate Unreferenced Elements	Select this check box to have Java code generated for unreferenced (declared in the schema but not used) elements as well.	Apache Axis
Support Wrapped Document/Literal Style	Use this check box to configure processing of "wrapped" document/literal, which is a document literal variation, that wraps parameters as children of the root element. – When this check box is cleared, no special treatment is applied to "wrapped" document/literal style operations. – When the check box is selected, a set of conditions is considered to decide whether top level elements are "unwrapped" and each component of the element should be treated as an argument of the operation. The following conditions are considered: – An input message consists of single part. – This single part is an element. – The element has the same name as the operation. – The element's complex type has no attributes. By default, the check box is selected.	Apache Axis
Status	View the information in this read-only field to track and improve discrepancies when configuring the code generation procedure.	All

Monitor SOAP Messages Dialog

Tools | Web Services | Axis | Monitor SOAP Messages

Use this dialog box to monitor SOAP messages testing the client side of an Apache Axis Web service.

Item	Description
Web context name	Use this drop-down list to specify the context name of the server that is used for running the Web module on your local Tomcat session.
Port Name	Use this drop-down list to specify the port of the SOAP TCP monitor servlet, which is retrieved from `web.xml` descriptor settings.

Show Deployed Web Services Dialog

Tools | Web Services | Show Deployed Web Services

Use this dialog box to have a list of all Web services deployed at the current server host displayed.

Item	Description
Context name	In this text box, type the application context under which you want to see the deployed Web services.
Status	This read-only field shows information on the status of the target server and hints in the format of the data specified in the Context name text box.

XML-Java Binding Reference

In this section:

- Generate Java from XML Schema Using JAXB Dialog
- Generate XML Schema from Java Using JAXB Dialog
- Generate Java Code from XML Schema Using XmlBeans Dialog
- Generate Instance Document from Schema Dialog
- Generate Schema from Instance Document Dialog

Generate Java from Xml Schema using JAXB Dialog

Tools | JAXB | Generate Java Code From XML Schema Using JAXB

Use this dialog box to configure generation of Java code stubs based on an XML Schema via the JAXB data binder.

- This functionality is provided via the WebServices bundled plugin, which is enabled by default. If not, enable it as described in the section Enabling and Disabling Plugins .
- The menu item and the dialog box are available when the file opened in the active editor tab contains an XML Schema.

Item	Description
JAXB Schemas / wsdl / dtd path	In this field, specify the file to be used as the generation basis. By default, the field shows the full path to the current file. To use another Schema, click the Browse button and choose the desired file in the Select XML Schema File for JAXB Generation dialog box, that opens.
Output path	From this drop-down list, select the module source directory to place the generated Java code stubs in.
Package prefix	Use this drop-down list to specify the package to place the generated Java files in.
Generate package level annotations	Do one of the following: - Select this check box to have a `package-info.java` with annotations generated. - Clear this check box to have annotations internalized into other generated classes.
Mark generated code with 'generated' annotation	Select this check box to have generated code supplied with the `javax.annotation.Generated` annotations.
Make generated file read-only	Select this check box to force the XJC binding compiler to mark the generated Java source code as read-only.
Add necessary libraries in order for generated code compile and work	Select this check box to have additional JAXB client libraries automatically added to the classpath of the module where the generated source code will be placed. These are libraries the generated stubs code depend on.
Do not generate header	Select this check box to pass `no-header` parameter to the corresponding command.
Add external binding file/dir	Select this check box to specify an external binding file or an output directory where the generated file is located.
Status	View the information in this read-only field to track and improve discrepancies when configuring the generation procedure.

Generate XML Schema From Java Using JAXB Dialog

Tools | JAXB | Generate XML Schema From Java Using JAXB

editor | context menu of a class name | Web Services | Generate XML Schema From Java Using JAXB

Use this dialog box to configure XML Schema generation based on the existing Java code.

Note This functionality is provided via the WebServices bundled plugin, which is enabled by default. If not, enable it as described in the section Enabling and Disabling Plugins .
2. The menu item and the dialog box are available when a Java class is opened in the active editor tab.

Item	Description
Class Name	This read-only field shows the name of the class to base the XML Schema generation on.
Include parameter and return type of the following methods	– When the check box is cleared, the parameter and return type of all the class methods will be reflected in the generated Schema. – Select this check box to have a list of the class methods displayed and specify the methods to be involved in Schema generation. Selecting this check box enables the Parameter / return type of the following method and the Add to JAXB generation controls.
Parameter / return type of the following method	This read-only field shows a list of all the methods of the current class.
Add to JAXB generation	Select this check to have the parameter/return type of the corresponding method involved in Schema generation.
Status	View the information in this read-only field to track and improve discrepancies when configuring the Schema generation procedure.

Generate Java Code from XML Schema using XmlBeans Dialog

Tools | XmlBeans | Generate Java Code From XML Schema Using XmlBeans

Use this dialog box to configure generation of Java code stubs based on an XML Schema via the XmlBeans data binder.

Getting access to the dialog box

1. This functionality is provided via the WebServices bundled plugin, which is enabled by default. If not, enable it as described in the section Enabling and Disabling Plugins .
2. The menu item and the dialog box are available when the file opened in the active editor tab contains an XML Schema.

Item	Description
Schema path	In this field, specify the file to be used as the generation basis. By default, the field shows the full path to the current file. To use another Schema, click the Browse button … and choose the desired file in the Select XML Schema / Wsdl File for generation dialog box, that opens.
Output path	Use this field to specify the name of the `.jar` to place the generated and complied Java code in. By default, IntelliJ IDEA suggests to create a new `types.jar` . To overwrite an existing `.jar` , click the Browse button … and choose the desired `.jar` in the dialog that opens .
Add necessary libraries in order for generated code compile and work	Select this check box to have additional XmlBeans libraries automatically added to the classpath of the module where the generated source code will be placed. These are libraries the generated stubs depend on.
Status	View the information in this read-only field to track and improve discrepancies when configuring the generation procedure.

Generate Instance Document from Schema Dialog

Tools | XML Actions | Generate XML Document from XSD Schema

In this dialog box, specify the options for generating an XML file based on the XSD (XML Schema Definition) Schema that describes its structure.

The menu item and the dialog box are available when the file opened in the active editor tab contains an XML Schema.

Item	Description
Schema path	In this field, specify the location of the `.xsd` Schema file to be used as the basis for XML document generation. By default, the field shows the full path to the current file. To use another Schema, click the Browse button ... and select the desired file in the dialog that opens .
Instance document name	In this text box, specify the name of the output XML file to be generated. By default, the generated XML file will inherit the name of the source Schema and will have the `.xml` extension. If you type another name for the document to be generated, make sure the extension is correct. The default location for the generated document is the directory where the source `.xsd` Schema file resides. To specify another location, click the Browse button ... and select the desired path in the dialog that opens .
Element name	In this drop-down list, specify the local name of the global element to be used as the root of the generated document.
Enable restriction check	Select this check box to have IntelliJ IDEA take restriction particles into consideration (if the specified Schema uses any).
Enable unique check	Select this check box to have IntelliJ IDEA take uniqueness particles into consideration (if the specified Schema uses any).
Status	View the information in this read-only field to track and improve discrepancies when configuring the generation procedure.

Generate Schema from Instance Document Dialog

Tools | XML Actions | Generate XSD Schema from XML File

In this dialog box, specify the options for generating an XSD (XML Schema Definition) Schema that describes the structure of the desired XML file.

The menu item and the dialog box are available when an XML document is opened in the active editor tab.

Item	Description
Instance document path	In this text box, specify the full path to the XML document to be used as the basis for Schema generation. By default, the field shows the full path to the current file. To use another XML document, click the Browse button ... and select the desired file in the dialog that opens .
Result Schema file name	In this text box, specify the name of the output file for the Schema to be generated. By default, the generated XSD Schema file will inherit the name of the instance document used and will have the `.xsd` extension. If you type another name for the Schema to be generated, make sure the extension is correct. The default location for the generated Schema is the directory where the source XML instance document resides. To specify another location, click the Browse button ... and select the desired path in the dialog that opens .
Design type	Use this drop-down to specify how to declare elements and complex types. The available options are: – Local elements / global complex types – Local elements / types – Global elements / local types
Detect simple content types	From this drop-down list, choose the type to render leaf text, which should be distinguished from the text used. The available options are: – String – Smart
Detect enumeration limit	In this text box, type the number of occurrences to cause the Schema enumeration appearance. To suppress Schema enumeration, specify 0.

Table Editor

From the Database tool window (for any table within a DB data source):

– on the toolbar (if the toolbar is not currently hidden)

– Table Editor from the context menu

– F4

Use the Table Editor to manipulate the table data, and to perform other, associated tasks.

There are two tabs in the Table Editor:

– Data. On this tab, the table you are working with is shown.

– DDL. Initially, the `CREATE TABLE` statement used to create the table is shown on this tab. You can edit the statement and then run it (on the toolbar or Ctrl+Shift+F10). Use or Ctrl+F5 to regenerate the `CREATE TABLE` statement for the current state of the table in the database.

Below, the features of the Data tab are discussed.

– Toolbar controls, context menu commands for data cells and keyboard shortcuts

– Using the table header row: sorting data, reordering and hiding columns

See also, Working with the Table Editor .

Toolbar controls, context menu commands for data cells and keyboard shortcuts

Most of the functions on the Data tab are accessed by means of controls on the toolbar, context menu commands for the data cells, and associated keyboard shortcuts.

Item	Shortcut	Description
and		These icons and corresponding commands are for switching between the result set pages, i.e. the pages that show the table data. A fixed number of rows shown simultaneously is referred to as a result set page. If this number is less than the number of rows in the table, only a subset of all the rows is shown at a time. In such cases, you can use and to switch between the subsets. (If all the rows are currently shown, these icons and the corresponding commands are inactive.) The result set page size is set on the Database page of the Settings dialog.
First Page		Use this icon or command to switch to the first of the result set pages to see the first series of rows.
Previous Page	Ctrl+Alt+Up	Use this icon, command or shortcut to switch to the previous result set page to see the previous series of rows.
Next Page	Ctrl+Alt+Down	Use this icon, command or shortcut to switch to the next result set page to see the next series of rows.
Last Page		Use this icon or command to switch to the last of the result set pages to see the last series of rows.
Reload Page	Ctrl+F5	Use this icon, command or shortcut to refresh the current table view. Use this function to: – Synchronize the data shown with the actual contents of the database. – Apply the Result set page size setting after its change.
Add New Row	Alt+Insert	Use this icon, command or shortcut to add a new row to the table. Complete entering a value into a cell by pressing Enter . To save the new row, select Submit New Row from the context menu or press Ctrl+Enter . See also, Adding a row.
Delete Selected Rows	Ctrl+Y	Use this icon, command or shortcut to delete the selected row or rows. Rows are selected by clicking the cells in the column where the row numbers are shown. To select more than one row, use mouse clicks in combination with the Ctrl key.
Auto-commit		Use this check box to turn the autocommit mode for the database connection on or off. In the autocommit mode, each SQL statement is executed in its own transaction that is implicitly committed. Consequently, the SQL statements executed in this mode cannot be rolled back. (Each change of a value in the table, or adding or deleting a row results in executing an SQL statement.) If the autocommit mode is off, transactions are committed or rolled back explicitly by means of the commit or rollback command. Each commit or rollback starts a new transaction which provides grouping for a series of subsequent SQL statements. In this case, the data manipulations in the transaction scope are committed or rolled back all at once when the transaction is committed or rolled back.
Commit		If the autocommit mode is off, use this icon or command to commit the current transaction.
Rollback		If the autocommit mode is off, use this icon or command to roll back the current transaction.
Cancel Query	Ctrl+F2	Use this icon or shortcut to terminate execution of the current query.
Tab S...(TSV) Data Extractor		Use this button or command to open a menu in which you can select an output format for your data. The Configure CSV Formats command in that menu opens the CSV Formats Dialog that lets you manage the delimiter-separated values formats (e.g. CSV, TSV).
View Query		Use this button to view the query which was used to generate the current table view. To close the pane where the query is shown, press Escape .
		This icon provides access to the following commands: – Transpose. Use this command to turn the transposed table view on or off. (In the transposed view, the rows and columns are interchanged. So, the rows are shown as columns and vice versa.) – Reset View. Use this command to restore the initial table view after reordering or hiding the columns, or sorting the data. – Sort via ORDER BY. Use this command to turn the corresponding option on or off. If the Sort via ORDER BY option is on, all the sorting operations that you perform are reflected in the corresponding `SELECT` statement

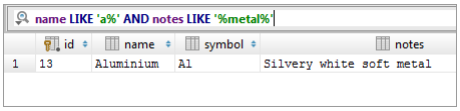
(an `ORDER BY` clause is added or modified) which is executed immediately. As a result, the data for the whole table is sorted by the corresponding database system.

Don't turn this option on if you want to keep interactions with the database to a minimum (e.g. when the table is very big or the database connection is "slow").

If this option is off, the data is sorted "locally" by IntelliJ IDEA and only for the rows currently shown.

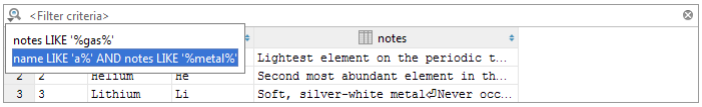
- Row Filter. Use this command to show or hide the box in which you can specify filtering conditions for your table.
- Settings. Use this command to open the Database page of the Settings dialog to view or edit the settings for the Database, Hibernate and JPA consoles, the Table Editor and the Database tool window.

In this box, specify filtering conditions for the table. (If the box is not currently shown, click on the toolbar and select Row Filter.)
The filtering conditions are specified as in a `WHERE` clause but without the word `WHERE`, e. g. `name LIKE 'a%' AND notes LIKE '%metal%'`. Within the `LIKE` expressions, the SQL wildcards can be used: the percent sign (`%`) for zero or more characters and underscore (`_`) for a single character.



To apply the conditions currently specified in the box, press `Enter`. To cancel filtering, delete the contents of the filter box and press `Enter`.

To reapply a memorized filter, click and select the filter in the list. See also, Filter history size.



To cancel filtering and hide the filter box, click .

Clone Row	`Ctrl+D`	Use this command or shortcut to create a copy of the selected row.
Edit Value	`F2`	Use this command or shortcut to start editing a value in the selected table cell. (Alternatively, you can double-click the cell or simply start typing.) To save the modified value, press `Enter`. To cancel editing, press `Escape`. See also, Modifying cell contents.
Quick Documentation	`Ctrl+Q`	Use this command or shortcut to open the quick documentation view. To close the view, press `Escape`. For more information, see Using the quick documentation view.
Go To | Row	`Ctrl+G`	Use this command or shortcut to switch to a specified row. In the dialog that opens, specify the row number to go to.
Go To | Related Data	`F4`	Use this command or shortcut to switch to a related record. The command options are a combination of those for Go To | Referenced Data and Go To | Referencing Data. The command is not available if there are no related records.
Go To | Referenced Data	`Ctrl+B`	Use this command or shortcut to switch to a record that the current record references. If more than one record is referenced, select the target record in the pop-up that appears. The command is not available if there are no referenced records.
Go To | Referencing Data	`Alt+F7`	Use this command or shortcut to see the records that reference the current record. In the pop-up that appears there are two categories for the target records: – First Referencing Row. All the rows in the corresponding table will be shown and the first of the rows that references the current row will be selected. – All Referencing Rows. Only the rows that reference the current row will be shown. The command is not available if there are no records that reference the current one.
Filter by		Use this command to access quick filtering options. The options include those for the current column name and depend on the value in the current cell.
Dump Data | To Clipboard		Use this command to copy the table data onto the clipboard.
Dump Data | To File		Use this command to save the table data in a file. In the dialog that opens, specify the location and name of the file.
Save LOB		Use this command to save the large object (LOB) currently selected in the table in a file.

Using the table header row: sorting data, reordering and hiding columns

Use the cells in the header row (i.e. the row where column names are shown) for:

- Sorting data
- Reordering columns
- Hiding and showing columns

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: name . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance: name 1. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: .

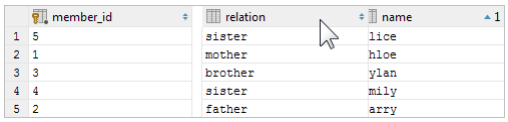
Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: .

Here is an example of a table where data are sorted by two of its columns.

	id	name ▲2	relation ▲1
1	3	Dylan	brother
2	6	Jack	brother
3	1	Harry	father
4	2	Chloe	mother
5	5	Alice	sister
6	4	Emily	sister

To restore the initial "unsorted" state for the table, click and select Reset View. See also, Sort via ORDER BY.

To reorder columns, use drag-and-drop for the corresponding cells in the header row. To restore the initial order of columns, click and select Reset View.

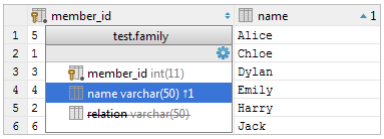


To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
   - Right-click any of the cells in the header row and select Column List.
   - Press `Ctrl+F12`.

   In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

To show all the columns, click and select Reset View.

See also, Using the Structure view to sort data, and hide and show columns.

Android Reference

In this part:

- Android Layout Preview Tool Window
- Android Monitor
- Choose Device Dialog
- Asset Studio
- Create Android Virtual Device Dialog
- Create Layout Dialog
- Designer Tool Window
- Generate Signed APK Wizard
- Inline Android Style Dialog
- New Android Component Dialog
- New Resource Directory Dialog
- New Resource File Dialog

Android Layout Preview Tool Window

The tool window appears when you open a layout definition file for editing in the manual mode . It allows you to adjust the appearance of the layout preview and to emulate different configurations. This helps you adjust your application to different Android platforms, devices, orientations, dock modes, locales, etc.

In this section:

- Toolbar
- Controls

Toolbar

Use the controls in this area to adjust the appearance of the layout preview.

	ItemTooltipDescription	
	Zoom to Fit	Toggle this button to have IntelliJ IDEA compress or expand the preview so it fits the target screen size .
	Reset Zoom to 100%	Click this button to have IntelliJ IDEA reset the zoom to preview the actual size.
	Zoom In	Click this button to have IntelliJ IDEA expand the preview.
	Zoom Out	Click this button to have IntelliJ IDEA compress the preview.
	Jump to Source	Click this button to switch to the Text tab where you can edit the application layout in the source `.xml` file.
	Refresh	Click this button to have IntelliJ IDEA update the preview so that it reflects on-the-fly changes to the current layout definition.
	Save Screenshot	Click this button to take a screenshot of the application preview. Note This button is only available from the Text tab of the layout preview window.
	Options	Click this button to configure the appearance and behavior of the tool window. – Hide for non-layout files: select this option to have IntelliJ IDEA temporarily close the tool window when the focus in the editor switches to a non-layout file. As soon as the same or another layout definition file is in focus, the tool window re-appears automatically. – Include Device Frames (if available): select this option to make the preview look like it is going to appear on the actual device. – Show Lighting Effect: select this option to display lighting effects to make the preview look more natural. Note This option is only available if the Include Device Frames option is selected.

Controls

Use the controls in this area to preview your layout in different configurations.

	ItemTooltipDescription	
	Configuration to render this layout in the IDE	From this drop-down list, select a layout configuration that you want to preview and edit, create a new layout configuration, or select different preview options. The available options are: – Create Landscape Variation: select this option to create a landscape version of your layout. The corresponding layout definition file will be generated in the `res\layout-land` folder. Once this variation is created, this menu option will be replaced with the Switch to layout-land option that opens the `layout-land\<layout_file_name>.xml` file for editing. – Create layout-xlarge Variation: select this option to create a variation of your layout for an extra large screen size (at least 960x720 dp). The corresponding layout definition file will be generated in the `res\layout-xlarge` folder. Once this variation is created, this menu option will be replaced with the Switch to layout-xlarge option that opens the `layout-xlarge\<layout_file_name>.xml` file for editing. – Switch to layout: this option is only available if you have created multiple layout versions. Select it to return to the original layout definition file. – Create Other: select this option to create another variation of your layout. In the Select Layout Directory dialog that opens, specify the folder where the layout definition will be stored and select resource qualifiers that determine a specific device configuration. Select the relevant qualifier and click ». Then specify the value of the qualifier in the dialog box that opens. The qualifier is added to the Chosen qualifiers list. – Preview Representative Sample select this option to display multiple device configurations and preview the layout on the most important screen sizes. – Preview All Screen Sizes: select this option to display multiple device configurations and preview the layout on all available screen sizes. – Preview All Locales: select this option to preview the layout in all locales where your application is going to be used. – Preview Right-to-Left Layout: select this option to preview the layout in both directions (left-to-right and right-to-left) side by side. – Preview Android Versions: select this option to preview the layout on all installed Android API versions. – Preview Included: select this option to preview your layout nested in another layout. This option is only available if the current layout is included into another layout. – Preview Layout Versions: select this option to display multiple device configurations and preview the layout in all available variations. – None: select this option to return to the default view. – Toggle Layout Mode: select this option to switch between different preview options.
	The virtual device to render the layout with	From this drop-down list, select a virtual or a physical device to preview what your application will look like on this device. To add a new virtual device, select Add Device Definition and configure an emulator in the Android Virtual Device (AVD) Manager that opens (for instructions refer to Managing Virtual Devices).
	Go to next state	From this drop-down list, select the preview orientation (portrait or landscape), the UI mode (Normal, Car Dock, Desk Dock, Television, Appliance) and switch between the Night (dimmed screen) and Not Night (standard brightness) modes. For details on UI modes refer to UiModeManager .
	Theme	Click this button to select a theme from the Select Theme dialog.
	N/A	Click this button to associate the layout with an activity. Select Associate with <activity_name> to associate it with the current activity, or

		Associate with Other Activity to display a list of available activities to select from.
	Locale to render layout with in the IDE	From this drop-down list, select an existing locale or add a locale for your application. A locale is a combination of the target country and language to have the dates and some other data presented in accordance with the local rules and preferences. You can also preview the layout in all available locales and in both directions (left-to-right and right-to-left).
	Android version to use when rendering layouts in the IDE	From this drop-down list, select an API version or use the Automatically Pick Best option to render the layout using the most suitable Android version. You can also preview the layout on all installed Android API versions.

Android Monitor

View | Tool Windows | Android Monitor

In this tool window, you can view and analyze the system debug output when running or debugging Android applications.

The tool window consists of a toolbar and the following nested tabs:

- Logcat tab
- Monitors tab that includes the following performance monitors :
  - Memory
  - CPU
  - Network
  - GPU

Select a physical or a virtual device from the devices drop-down list, and a process running on the selected device from the processes drop-down list. The information in each tab is filtered in accordance with your selection.

Toolbar

The toolbar that is common for all nested tabs, provides quick access to the following actions:

Icon	Tooltip	Description
	Screen Capture	Click this button to make a screenshot of the virtual or the connected physical device where the application is running. You can rotate the screenshot and add some visual effects in the dialog that opens, and save the capture.
	Screen Record	Click this button to start video recording of the application output on a physical device. This button only becomes available when you connect a physical Android device to your computer.
	System Information	Click this button to view system information on the selected process provided by the Dumpsys tool. Logs open as text files in separate editor tabs. Select which type of information you want to view from the popup menu that opens: – Activity Manager State: select this option to display system output of the Activity Manager - a component responsible for managing a stack of the application's activities. – Package Information: select this option to display system information on the application package. – Memory Usage: select this option to display detailed information on the application memory usage in kB. – Memory use over time: select this option to display detailed information on the system memory usage aggregated in the last 24 hours and the last 3 hours, and the current memory usage statistics. – Graphics State: select this option to display detailed application's graphics acceleration info.
	Terminate Application	Click this icon to stop the application execution.

Logcat tab

This tab shows all system debug output messages related to the selected process on the selected device.

Toolbar

Use the toolbar controls and buttons to configure the scope and the presentation of log data, and to navigate through the log.

Item	Tooltip and shortcut	Description
Log Level	N/A	From this drop-down list, select the priority of log messages to be displayed. The available options are: – Verbose – Debug – Info – Warn – Error – Assert
	Find	Use this text box to search through the list of messages. As you type a search string, the messages that match the search pattern are displayed with the matching character strings highlighted. To finalize the search, press `Enter`. Search patterns are stored in the search history list. To clear the search history, click the button.

Filters	N/A	From this list, select an existing filter configuration, or create a new one.
		A filter configuration is a set of filtering parameters. The use of filter configurations provides more flexible control over the type and amount of log data displayed than just specifying the information type by choosing a message priority in the Log level drop-down list.
		– To display messages related to the selected process, select Show only selected application.
		– If you want full log data to be displayed, select No Filters from the drop-down list.
		– To apply a filter, select it from the drop-down list.
		– To create a new filter configuration, select Edit Filter Configuration. In the Create New Logcat Filter dialog box that opens, click the Add toolbar button + and specify the following filtering parameters:
		  – Log Tag: use this parameter if you only want messages from a certain system component to be displayed. Type a regular expression to specify the tag that indicates the relevant system component, such as ActivityManager, AudioService, etc., or a user-defined tag. For more details, see Filtering Log Output .
		  – Log Message: use this parameter if you only want messages that contain certain elements or character strings to be displayed. Type a regular expression that defines the character string to be detected.
		  – Package Name: use this parameter if you only want messages that refer to a specific Java package (class path) to be displayed.
		  – PID: use this parameter if you only want messages that refer to a specific process (process ID) to be displayed.
		  – Log Level: use this parameter if you only want messages with a certain priority level to be displayed.

- To update a filter, select Edit Filter Configuration, then in the Create New Logcat Filter dialog that opens select a filter from the list and edit the filter values.
- To remove a filter configuration from the list, choose Edit Filter Configuration. In the Create New Logcat Filter dialog box that opens, select the filter and click the Delete toolbar button —.

	Clear logcat	Click this button to remove log data from previous sessions on the selected device.
	Scroll to the end	Click this button to move the caret to the last line of the console output.
	Up the Stack Trace `Ctrl+Alt+Up`	Click this button to navigate up in the stack trace and have the cursor jump to the corresponding location in the source code.
	Down the Stack Trace `Ctrl+Alt+Down`	Click this button to navigate down in the stack trace and have the cursor jump to the corresponding location in the source code.
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Print	Click this button to print the logs.
	Restart	Click this button to restart logging.

Monitors tab

The Monitors tab displays different performance monitors and their toolbars.

Memory

The Memory monitor shows in real-time how your application allocates memory. It displays available and used memory in a graph, which is good for testing purposes and helps detect whether performance issues are related to garbage collection events:

Toolbar

Item	Tooltip	Description
	Initiate GC	Click this button to start garbage collection, whereupon you can examine the amount of heap memory the selected process uses (see Viewing heap usage).
	Dump Java Heap	Click this button to dump the contents of the Java heap memory to a file.
	Start Allocation Tracking	Click this button to start recording your application's memory allocations. Then interact with your application and click the same button to stop allocation tracking. A pane with the recorded data will open as a separate editor tab. Allocation tracking is helpful to identify where similar object types are allocated and deallocated over a short period of time, and to detect places in your code that may cause inefficient memory usage.

CPU

The CPU monitor shows in real time how much CPU is being used in a graph.

Toolbar

Item	Tooltip	Description
	Start Method Tracing	Click this button to invoke profiling of a method, see Starting method profiling . Interact with the application, and then click the same button to stop method tracing. A pane with the recorded data will open as a separate editor tab. Method tracking is useful to track down performance issues.

Network

The Network monitor enables you to track your application network requests.

GPU

The GPU monitor displays how much time it takes to render the frames of a UI window.

Choose Device Dialog

Run | Run

Run | Debug

This dialog box opens when you start a run or a debug session with the manual selection of the target device specified in the run/debug configuration (i.e. the Show chooser dialog option is selected in the Run/Debug Configuration: Android Application dialog). It shows the list of all currently running devices, both physical and virtual.

Use this dialog box to appoint a running device, or to launch a virtual device.

Item	Description
Choose a running device	Select this option to choose a running device from the list below. The following information is provided on each device: – Device: the device name. – Serial Number: the device serial number assigned to it by the manufacturer. – State: the device current state (Online or Offline). – Compatible: shows whether the device is compatible with the application settings.
Launch emulator	Select this option to launch a virtual Android device. To create a new device, click the Browse button to launch the Android Virtual Device Manager.
Use same device for future launches	Select this checkbox to use the selected device for the future run and debug sessions.

Asset Studio

To access the Asset Studio wizard, in the Project tool window , right-click the `res` folder and select New | Image Asset from the popup menu.

The Asset Studio wizard is a convenient tool that allows you to create icons for your Android applications. It creates multiple icons for different screen resolutions and lets you see a live preview in the process of creating. You can create icons using your own images, clipart images, or text, configure the background shape, the colors, the fonts, etc.

– Asset Studio. Page 1

– Asset Studio. Page 2

Asset Studio. Page 1

Use this page to select the image source, adjust the original image to the icon size and aspect ratio, modify shapes and colors, etc.

Item	Description
Asset Type	Select the type of icon you want to create from the drop-down list. The available options are: – Launcher Icons – Action Bar and Tab Icons – Notification Icons
Foreground	Select one of the following sources for your icon: – Image – Clipart – Text
Image file	This field is only available if Image is selected as the icon source. Specify the path to the image file, or click the Browse button and select the image file in the dialog that opens.
Clipart	This control is only available if Clipart is selected as the icon source. Click the Choose button to select the foreground image from the clipart collection.
Text	This control is only available if Text is selected as the icon source. Enter the text and select the font from the Font drop-down list.
Trim surrounding blank space	Select this option if you want to contract the blank space around your image and adjust the foreground to the icon size.
Additional padding	Drag the slider if you want to increase/decrease the amount of blank space around the foreground image.
Foreground scaling	This control is only available if Launcher Icons is selected as the asset type. The image that servers as the source for your icon may not have the same aspect ratio as the icon itself. You can adjust it be selecting one of the following options: – Crop: the foreground image will be cropped so that it takes the full space of the parent element. – Center: the foreground image will remain unchanged and will be placed in the middle of the parent element.
Shape	These controls are only available if Launcher Icons is selected as the asset type. Use these controls to add a shaped frame to your icon. The following options are available: – None – Square – Circle
Background color	This control is only available if Launcher Icons is selected as the asset type. Click the color block to open the Select Color dialog where you can choose the color that will fill the background if you have selected Square or Circle above as the icon shape.
Foreground color	This control is only available if Launcher Icons is selected as the asset type, and Clipart or Text is selected as the icon source. Click the color block to open the Select Color dialog where you can choose the color for the foreground image.
Theme	This control is only available if Action Bar and Tab Icons is selected as the asset type. Select a theme for you icon from the drop-down list.
Resource name	Enter the name for the new drawable resource.

Asset Studio. Page 2

Use this page to specify the target module for the new drawable resource.

Item	Description
Res Directory	This field shows the path to your resource directory.
Output Directories	This pane shows the target folders for the new drawable resource in different output formats.

Captures Tool Window

View | Tool Windows | Captures

In this topic:

– Overview
– Allocation Tracker
– Memory viewer
– Context menu of a selection

Overview

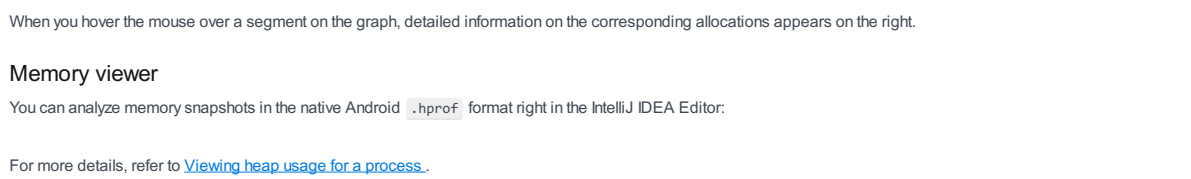
This tool window contains a list of system captures you've made when debugging your Android application:

– Allocation Tracking captures
– Heap Snapshots
– Method Tracing captures
– System Information captures

Click a file to open it as a separate tab in the Editor.

Allocation Tracker

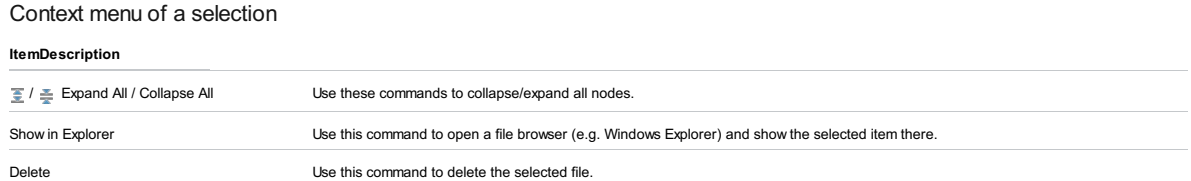
IntelliJ IDEA lets you analyse memory allocations that your application uses. In addition to a table view, you can click the icon in the toolbar of the Allocations tab in the Editor to open a visual representation of your application's memory allocations:



When you hover the mouse over a segment on the graph, detailed information on the corresponding allocations appears on the right.

Memory viewer

You can analyze memory snapshots in the native Android `.hprof` format right in the IntelliJ IDEA Editor:



For more details, refer to Viewing heap usage for a process .

Context menu of a selection

Item	Description
/ Expand All / Collapse All	Use these commands to collapse/expand all nodes.
Show in Explorer	Use this command to open a file browser (e.g. Windows Explorer) and show the selected item there.
Delete	Use this command to delete the selected file.

Create Android Virtual Device Dialog

This dialog box opens when you first launch a Run/Debug session for an Android module, Emulator is selected as the target device on module creation, and you have no Android Virtual Devices configured.

Use this dialog box to create new emulators.

Item	Description
Name	In this text box, type the name of the new emulator.
Target	From this drop-down list, select the Android platform to have the application built against.
Skin	From this drop-down list, choose a skin to use for controlling screen dimensions (optional).
Abi Type	From this drop-down list, select the Application Binary Interface (ABI) type.
SDCard	If you want to simulate the presence of an SD card in the virtual device, you can create an SD card image by generating a dedicated file and specifying the virtual SD card size (optional).

Create Layout Dialog

Project Tool Window | context menu of a package with the application classes | New - Android Component

The dialog box opens when you create a component of the activity or fragment type, and select the Create layout file check box.

In this dialog box, specify the layout name, its root element, and the set of qualifiers to be included in this layout.

Item	Description
File name	In this text box, specify the name of the layout definition file associated with the new component. Only lowercase letters and numbers are supported.
Root element	In this text box, specify the root element of the layout. Press `Ctrl+Space` to get a list of available values. The value must be of the View or ViewGroup type, learn more at http://developer.android.com/guide/topics/resources/layout-resource.html .
Directory name	In this text box, specify the folder where the layout definition will be stored relative to the `res` folder. – If your application does not need to be compatible with various Android devices and, therefore, no multiple screens support is required, accept the default `layout` subfolder. – To provide alternative resources , specify the resource qualifiers that determine a specific device configuration. Move the relevant qualifiers from the Available qualifiers list to the Chosen qualifiers list and specify their values. IntelliJ IDEA appends all the selected qualifiers to the Directory name field with a dash character as separator. For details, see Creating Resources .
Available qualifiers	From this list, select the resource qualifiers that determine a specific device configuration. Select the relevant qualifier and click ». Then specify the value of the qualifier in the dialog box that opens. The qualifier is added to the Chosen qualifiers list.

Designer Tool Window

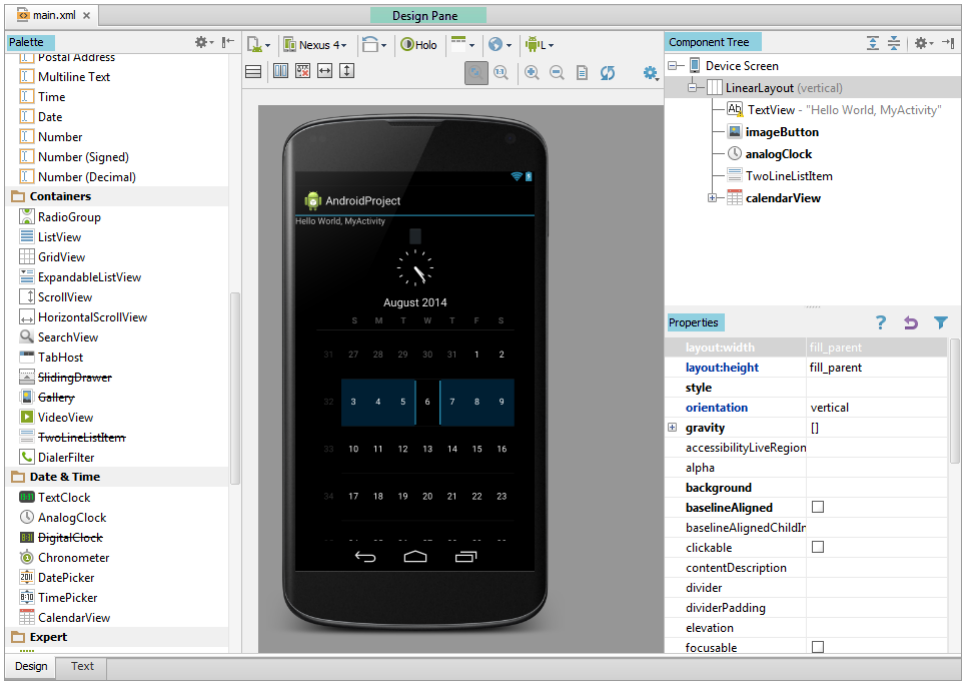
Use this dedicated tool window to build the design of your application without editing the layout definition files manually, and check how the application design is rendered in various target environments without running the application on any physical or virtual devices.

> **Note** Alternatively, edit the layout definition files manually, possibly using the Android-specific refactoring provided by IntelliJ IDEA, and preview the changes that are immediately reflected in the dedicated Preview tool window, where you can adjust the layout to various platforms and devices. To switch to the manual mode, click the Text tab or choose Go To Declaration from the context menu in the Design pane or in the Component Tree.

A layout defines the user interface of an activity or an app widget (fragment). Layouts are declared in XML resource definition files. See Creating Resources for instructions on how to create resource folders and resource definition files.

The Android UI designer consists of the following panes:

- Design Pane
- Component Tree
- Properties
- Palette



Design Pane

The Design pane is located in the central part of the UI Designer (assuming the default tool window layout). When you open a layout definition file for editing, the pane appears in the editor tab by default. If you are editing the layout definition file manually and then switch to the visual mode by clicking the Design tab, the pane opens in the tab where the edited layout definition file is opened,

To toggle between the manual and visual editing modes, use the Text and Design tabs in the bottom of the pane/editor:

The pane shows a rectangular canvas that is synchronized with the current layout definition file and with the Component Tree view, so any changes to the canvas are reflected there accordingly.

To add a component to the canvas, do one of the following:

- Select the required element in the Palette pane and drag and drop it to the canvas in the Design pane.
- Click the required element in the Palette pane and then click an area on the canvas.

Every component added in either way is also added to the Component Tree and is declared in the layout definition file.

You can also specify the most essential properties for a component right in the canvas, without switching to the Properties pane. To do that, select the component in question, double click the selected area, and fill in the fields in the pop-up dialog box that opens.

The canvas has a context menu that provides access to the clipboard, layout actions, refactoring, and more.

You can view what the built layout will look like on various devices: configure the target environment emulation using the controls on the Controls .

For details on different configuration options available from the Design pane, refer to the following sections:

- Context Menu
- Toolbar
- Controls

Context Menu

The context menu is available from any area in the Design pane, both inside the canvas and outside it. All actions are synchronized with the Component Tree and the XML definition files, so all changes are immediately reflected in them. The same actions are also available from the context menu of the Component Tree pane.

Item	Shortcut	Description
Cut	Ctrl+X	Choose this option to cut the selected component.
Copy	Ctrl+C	Choose this option to copy the selected component to the clipboard.
Paste	Ctrl+V	Choose this option to insert the component from the clipboard inside the selected parent component in the canvas. Note This action is available only when a parent component is selected.
Delete	Delete	Choose this option to remove the selected component from the canvas and from the layout.
Select	N/A	Choose this option to select one of the following: – Select Parent: choose this option to select the parent component of the selected component. – Select Siblings: choose this option to select the components located on the same level as the selected component. – Select Same Type: choose this option to select the components of the same type as the selected component. – Select All: choose this option to select all components on the canvas. – Deselect All: choose this option to deselect all selected components.
Morphing	N/A	Choose this option to convert the selected component into a component of another type preserving the properties that are common for both types. When you choose this option, IntelliJ IDEA shows a list of compatible component types, that is, the types into which the selected component can be converted.
Save Screenshot	N/A	Choose this option to take a screenshot of the current layout.
Refactor	N/A	Choose this option to apply one of the available refactorings to the current XML layout definition file. The XML code will be optimized while the layout itself will not be affected. The list of available refactorings depends on the type of the selected component.
Go To Declaration	Ctrl+B	Choose this option to navigate to the definition of the selected component in the XML layout definition file. Note Choosing this option automatically brings you to the manual layout editing mode. To return to the design mode, switch to the Design tab in the bottom of the editor.

Toolbar

Use the controls in this area to adjust the appearance of the layout preview.

Item	Tooltip	Description
	Zoom to Fit	Toggle this button to have IntelliJ IDEA compress or expand the preview so it fits the target screen size .
	Reset Zoom to 100%	Click this button to have IntelliJ IDEA reset the zoom to preview the actual size.
	Zoom In	Click this button to have IntelliJ IDEA expand the preview.
	Zoom Out	Click this button to have IntelliJ IDEA compress the preview.
	Jump to Source	Click this button to switch to the Text tab where you can edit the application layout in the source `.xml` file.
	Refresh	Click this button to have IntelliJ IDEA update the preview so that it reflects on-the-fly changes to the current layout definition.
	Save Screenshot	Click this button to take a screenshot of the application preview. Note This button is only available from the Text tab of the layout preview window.
	Options	Click this button to configure the appearance and behavior of the tool window. – Hide for non-layout files: select this option to have IntelliJ IDEA temporarily close the tool window when the focus in the editor switches to a non-layout file. As soon as the same or another layout definition file is in focus, the tool window re-appears automatically. – Include Device Frames (if available): select this option to make the preview look like it is going to appear on the actual device. – Show Lighting Effect: select this option to display lighting effects to make the preview look more natural. Note This option is only available if the Include Device Frames option is selected.

Controls

Use the controls in this area to preview your layout in different configurations.

Item	Tooltip	Description
	Configuration to render this layout in the IDE	From this drop-down list, select a layout configuration that you want to preview and edit, create a new layout configuration, or select different preview options. The available options are: – Create Landscape Variation: select this option to create a landscape version of your layout. The corresponding layout definition file will be generated in the `res\layout-land` folder. Once this variation is created, this menu option will be replaced with the Switch to layout-land option that opens the `layout-land\<layout_file_name>.xml` file for editing. – Create layout-xlarge Variation: select this option to create a variation of your layout for an extra large screen size (at least 960x720 dp). The corresponding layout definition file will be generated in the `res\layout-xlarge` folder. Once this variation is created, this menu option will be replaced with the Switch to layout-xlarge option that opens the `layout-xlarge\<layout_file_name>.xml` file for editing. – Switch to layout: this option is only available if you have created multiple layout versions. Select it to return to the original layout definition file. – Create Other: select this option to create another variation of your layout. In the Select Layout Directory dialog that opens, specify the folder where the layout definition will be stored and select resource qualifiers that determine a specific device configuration. Select the relevant qualifier and click ». Then specify the value of the qualifier in the dialog box that opens. The qualifier is added to the Chosen qualifiers list. – Preview Representative Sample select this option to display multiple device configurations and preview the layout on the most important screen sizes.

- Preview All Screen Sizes: select this option to display multiple device configurations and preview the layout on all available screen sizes.
- Preview All Locales: select this option to preview the layout in all locales where your application is going to be used.
- Preview Right-to-Left Layout: select this option to preview the layout in both directions (left-to-right and right-to-left) side by side.
- Preview Android Versions: select this option to preview the layout on all installed Android API versions.
- Preview Included: select this option to preview your layout nested in another layout. This option is only available if the current layout is included into another layout.
- Preview Layout Versions: select this option to display multiple device configurations and preview the layout in all available variations.
- None: select this option to return to the default view.
- Toggle Layout Mode: select this option to switch between different preview options.

	The virtual device to render the layout with	From this drop-down list, select a virtual or a physical device to preview what your application will look like on this device. To add a new virtual device, select Add Device Definition and configure an emulator in the Android Virtual Device (AVD) Manager that opens (for instructions refer to Managing Virtual Devices).
	Go to next state	From this drop-down list, select the preview orientation (portrait or landscape), the UI mode (Normal, Car Dock, Desk Dock, Television, Appliance) and switch between the Night (dimmed screen) and Not Night (standard brightness) modes. For details on UI modes refer to UiModeManager .
	Theme	Click this button to select a theme from the Select Theme dialog.
	N/A	Click this button to associate the layout with an activity. Select Associate with <activity_name> to associate it with the current activity, or Associate with Other Activity to display a list of available activities to select from.
	Locale to render layout with in the IDE	From this drop-down list, select an existing locale or add a locale for your application. A locale is a combination of the target country and language to have the dates and some other data presented in accordance with the local rules and preferences. You can also preview the layout in all available locales and in both directions (left-to-right and right-to-left).
	Android version to use when rendering layouts in the IDE	From this drop-down list, select an API version or use the Automatically Pick Best option to render the layout using the most suitable Android version. You can also preview the layout on all installed Android API versions.

Component Tree

This pane shows a hierarchy of components in the current layout with the `Device Screen` root node. The pane is synchronized with the Design pane and the layout definition file, so any changes to them are reflected in the tree view on-the-fly.

For details on different configuration opions available from the Component Tree, refer to the following sections:

- Context Menu
- Toolbar

Context Menu

The context menu is available from any area in the Component Tree pane. All actions are synchronized with the Design Pane and the XML definition files, so all changes are immediately reflected in them. The actions available from the Component Tree context menu are identical with those of the Design Pane context menu .

Toolbar

Item	Tooltip and Shortcut	Description
	Expand All `Ctrl+NumPad Plus`	Click this button to expand all nodes in the Component Tree view.
	Collapse All `Ctrl+NumPad -`	Click this button to collapse all nodes in the Component Tree view.
	N/A	Click this button to open the context menu and configure the pane viewing mode .
	Hide `Shift+Escape`	Click this button to hide the pane.

Properties

In this pane, specify property values of the component that is currently selected in the canvas or in the component tree view. The most frequently used properties are displayed in bold, "expert" properties are shown in *italic*, properties whose values have been updated are highlighted in blue.

Toolbar

Item	Tooltip and Shortcut	Description
	Show documentation `Ctrl+Q`	Click this button to have IntelliJ IDEA display a brief documentation for the selected property from the Android reference .
	Restore default value `Delete`	For any newly added component, the default property values are set. After you edit a property value, this property is highlighted in blue. Click this button to discard the changes and restore the default value. When you edit the default value of an expert property, the property is also highlighted in blue and will remain in the list when the Show expert properties toggle button is released.
		Note The button is available only if you have re-defined the value of the selected property.

Show expert properties	By default, the pane shows only a standard set of properties, and the most frequently used ones are displayed in bold. However, you can have IntelliJ IDEA display the entire set of properties for the selected component to enable advanced component configuration. – Click this button on to have the pane display all properties that are defined for the selected component according to the specification. All "expert" properties are displayed in *Italic*. – Release this button to have only the standard set of properties displayed. If the value of an expert property has been updated, the property will still remain in the list.

Palette

The Palette pane contains a number of pre-built UI elements that you can drag-and-drop to the canvas to design the layout of your application. For detailed instructions refer to Designing Layout of Android Application .

For details on different opions available from the Palette pane, refer to the following sections:

– Toolbar

– UI Components

Toolbar

Item	Tooltip and Shortcut	Description
	N/A	Click this button to open the context menu and configure the pane viewing mode .
	Hide `Shift+Escape`	Click this button to hide the pane.

UI Components

This pane contains UI components which you can place in the canvas. The available components are grouped into categories. To hide/expand the contents of a group, click the group title. The following groups of UI components are available:

Components	Description
Layouts	Android Layouts definee the visual structure of your application's interface. They are a class that determines the way its children appear on the screen.
Widgets	Android widgets are interactive components in a user interface. You can select from a variety of widgets, such as buttons, check boxes, test fields, etc.
Text Fields	A text field allows the user to enter text into the application, such as name, password, number, etc.
Containers	Containers are composite views that are comprised of multiple other views.
Date & Time	This section provides predesigned UI elements related to date and time, such as different types of clocks, a date and time picker controls, etc.
Expert	This section contains extra predefined UI elements in addition to the most frequently used components.
Custom	In this section, you can choose and add UI components defined either in your project, or in the Android SDK. You can insert elaborate combinations of widgets, or even embed entire layouts into the current layout. IntelliJ IDEA checks for a resource definition of the appropriate type in the project and in the Android SDK. The following options are available: – include: click this item to choose the layout that you want to embed. In the Resources dialog box that opens, select the Project tab to pick a layout definition from within your project, or the System tab to search in the SDK. – fragment: click this item to choose the fragment that you want to embed. – requestFocus: click this item to include the `requestFocus` element in a `view` object. The `requestFocus` element gives its parent initial focus on the screen. – CustomView: click this item to choose a user-defined view that you want to embed.

Generate Signed APK Wizard

Build| Generate Signed APK

To deploy and run an Android Application package (.apk file) on physical devices, you need to sign it with you personal signature (certificate). Based on this signature, the Android system identifies the author of every deployed application. You do not need to apply for a personal signature to any authority, a signature generated by IntelliJ IDEA is quite sufficient.

Use the Generate Signed APK Wizard to have IntelliJ IDEA digitally sign Android Packages (`.apk` files) during package extraction. You can use previously created signature keys, create new ones in existing keystores, or create new keystores.

Alternatively, you can configure the `.apk` file as an artifact by creating an artifact definition of the Android application type with the Release signed package mode turned on. When IntelliJ IDEA builds the package in accordance with this definition, it signs the package automatically.

> **Note** An unsigned package is used when you want to test your application on an emulator. Unsigned packages can be extracted only through artifact definitions with the Release unsigned package mode turned on.
>
> You can also extract and sign debug packages. This is sufficient for testing and debugging applications but does not allow publishing them. Signing packages in the debug mode is available only through an artifact with the Debug package mode turned on.

In this section:

– Generate Signed APK Wizard. Specify Key and Keystore
– New Key Store Dialog
– Generate Signed APK Wizard. Specify APK Location

Generate Signed APK Wizard. Specify Key and Keystore

On this page of the Wizard, specify the key store file that contains the digital key to sign the package with. You can use an existing key, or create a new one in an existing keystore, or create a new keystore.

Item	Description
Key store path	In this text box, specify the location of the file where the key will be stored. Type the path manually or click the Choose existing button to choose the relevant file in the dialog that opens .
Create new	Click this button to open the New Key Store Dialog and configure a new key store and/or a release key to be generated.
Choose existing	Click this button to have your package signed with a key from an existing key store file. Choose the relevant key store file in the dialog that opens . Tip Later you can choose to use an existing key from this key store, or to have a new key generated in it.
Key store password	In this text box, type the password for the selected key store.
Key alias	In this text box, specify the alias to address the key to be used.
Key password	In this text box, specify the password to access the selected key.
Remember passwords	Select this check box to have IntelliJ IDEA automatically use the specified passwords to access the key store and the key.

New Key Store Dialog

This dialog box opens when you click the Create new button on the Generate Signed APK Wizard. Specify Key and Keystore page, or in the Android tab of the Artifact page.

In this dialog box, specify the data required for generating new keys and key stores.

Item	Description
Key store path	In this text box, specify the location of the file where the new key will be stored. Type the path manually, or click the Browse button to choose the file in the dialog that opens .
Password	In this text box, enter the password that will be used to access the key store.
Confirm	Re-type the password.
Alias	In this text box, specify the alias to address the new key.
Password	In this text box, enter the password that will be used to access the new key.
Confirm	Re-type the password.
Validity (years)	Use this spin box to specify how long the new key will be valid. Tip Set the validity period in accordance with the expected lifespan of your application .
Certificate	In this area, specify the following personal data: – First and Last Name – Organizational Unit – Organization – City or Locality – State or Province – Country Code

Generate Signed APK Wizard. Specify APK Location

On this, last, page of the Wizard, specify the output directory for the generated Android Package. Optionally, have IntelliJ IDEA obfuscate the application through integration with the ProGuard built-in tool.

Item	Description
Destination APK path	In this text box, specify the path where the generated `.apk` file will be stored. Type the path manually, or click the Browse button to choose the path in the dialog that opens .
Run ProGuard	Select this check box to have IntelliJ IDEA obfuscate the application through integration with the built-in ProGuard tool. If this option is selected, the default paths to the configuration files that are created automatically are displayed. For Android SDK tools revision 17 and higher, this option is selected by default. For already existing projects, this check box is cleared.

Inline Android Style Dialog

Refactor | Inline Style

<context menu of a selection> | Refactor | Inline Style

`Ctrl+Alt+N`

Use this dialog box to configure the Inline Style. This refactoring is opposite to the Extract Style refactoring and results in adding all attributes defined in a style to one or all components where this style is applied. This refactoring is also used when you need to merge a parent style with its inheritor .

Item	Description
Inline all references and remove the style	Select this option to have IntelliJ IDEA convert the attributes of the style into XML tags, add them to the definitions of all components where the style is used, and remove the style definition.
Inline this usage and keep the style	Select this option to have the refactoring applied to the current component only. Note When the refactoring is invoked from the style definition file , only the Inline all references and remove the style option is available and selected by default.
Refactor	Click this button to apply the changes immediately.
Preview	Click this button to preview the changes before applying in the Find tool window .

New Android Component Dialog

Project Tool Window | context menu of a package with the application classes | New - Android Component

In this dialog box, specify the class that implements the new Android component, the component type, and the title to be displayed to the user.

Item	Description
Name	In this text box, type the name of the class that implements the component.
Kind	In this drop-down list, specify the component type. The available options are: – AIDL : an Android Interface Definition Language (AIDL) interface used for interprocess communication. – Activity : implements a window where you place your UI to interact with the user. – Android Auto : lets you extend your application for use in vehicles. You can add either Media Service or Messaging Service activity. – Folder: creates a source root based on the activity you have selected for this component. – Fragment : represents a behavior or a part of user interface in an activity. – Google: lets you create an activity for Google maps and AdMob Ads activities. – Application : an Android package, i.e. an `.apk` archive that contains the contents of an Android app and the installer. – Service : represents an application's desire either to perform an operation without interacting with the user, or to supply functionality for other applications. – Other: lets you add the following components to your application: – Android Manifest File – Broadcast Receiver – Content Provider – Daydream – UI Component : lets you add custom views to you application. – Wear : lets you extend your application for use in Android wear. – Widget : lets you add different types of widgets for your application. – XML : lets you add different types .xml files for Android layouts and values.
Label	In this text box, type the title that will be displayed to the user.
Mark as startup Activity	Select this check box to have the new component of the Activity type displayed by default when the application starts. Note The check box is available only when the Activity component type is specified in the Kind drop-down list.
Create layout file	– Select this check box to have IntelliJ IDEA generate a stub of the related layout definition file, that is, the content view of the new activity or fragment. – If the check box is cleared, you will have to create a content view of the new activity or fragment manually. Note The check box is available only for the activity or fragment component types.

New Resource Directory Dialog

Project Tool Window | context menu of the res folder | New | Android resource directory

Use this dialog box to create a resource directory for your Android project.

Item	Description
Directory name	In this text box, specify the name for the new resource directory.
Resource type	From this drop-down list, select the application resource type that the new directory will contain.
Available qualifiers	From this list, select the resource qualifiers that determine a specific device configuration. Select the relevant qualifier and click ». Then specify the value of the qualifier in the dialog box that opens. The qualifier is added to the Chosen qualifiers list.

New Resource File Dialog

Project Tool Window | context menu of the res folder | New | Android resource file

Use this dialog box to create a resource file for your Android project.

Item	Description
File name	In this text box, specify the name for the new resource definition file.
Resource type	From this drop-down list, select the application resource type . Tip You can scroll through the list of resource types right from the File name text box by using the Up and Down keyboard keys.
Root element	This field is populated automatically depending on the selected resource type.
Directory name	In this text box, specify the folder where application resources will be stored. Tip IntelliJ IDEA can automatically compose folder name based on the resource type and the qualifier you select. For details, refer to Creating Resources .
Available qualifiers	From this list, select the resource qualifiers that determine a specific device configuration. Select the relevant qualifier and click ». Then specify the value of the qualifier in the dialog box that opens. The qualifier is added to the Chosen qualifiers list.

Flex Reference

In this section:

- Create AIR Descriptor Template Dialog
- Create HTML Wrapper Template Dialog
- New ActionScript Class Dialog
- New MXML Component Dialog
- Package AIR Application Dialog
- Runtime-Loaded Modules Dialog

Create AIR Descriptor Template Dialog

Use this dialog to configure and create an application descriptor template for your AIR (Desktop or Mobile) application.

During the compilation or packaging, the text in the `<content>` element of the template will be replaced with the name and extension (`.swf`) of the application file.

- AIR application descriptor
- Application properties
- Mobile options
- Android tab
- iOS tab

AIR application descriptor

Item	Description
File name	Specify the name of the descriptor template `.xml` file to be generated.
Folder	Specify the folder in which the generated descriptor template should be stored. Use … (`Shift+Enter`) to select the folder in the corresponding dialog .
AIR version	Specify (select or type) the target AIR version.

Application properties

Item	Description
ID	Specify the application ID.
Name	Specify the application name, that is, the text to be displayed as the application title.
Version	Specify the application version in the `x.x.x` format. Note that this isn't the AIR version .

Mobile options

Item	Description
Mobile platform	Select which platform (Android or iOS) the application descriptor is intended for.
Common options	Select or deselect the following screen options : – Auto-orient. If selected, corresponds to `<autoOrients>true</autoOrients>` . – Full screen. If selected, corresponds to `<fullScreen>true</fullScreen>` .

Android tab

Item	Description
Permissions	Use the check boxes to enable or disable the corresponding Android permissions .

iOS tab

Item	Description
Devices	Select the supported device family. If iPhone/iPod Touch and iPad are supported, select All.
High resolution	Select this option if your application should use the high screen resolution mode.

Create HTML Wrapper Template Dialog

Use this dialog to configure and create an HTML wrapper template for your Web-targeted ActionScript or Flex application.

The generated template (`index.template.html`) will contain a set of tokens, such as `${title}` , `${swf}` , etc. During the compilation, these tokens will be replaced with the appropriate values. For example, `${swf}` will be replaced with the `.swf` file name. The resulting `.html` wrapper file will have the same name as the `.swf` file.

See also, Using the SWF metadata tag to control HTML wrapper properties .

Item	Description
Create HTML wrapper template in the following folder	Specify the folder in which the HTML wrapper template files should be created. Use ... (`Shift+Enter`) to select the folder in the corresponding dialog .
Enable integration with browser navigation	Select this option to enable deep linking. Deep linking lets users navigate their interactions with the application by using the Back and Forward buttons in their browser.
Check Flash player version	If you select this option, the compiled application will check for the correct version of Flash Player.
Express install	If you select this option, the application will run an SWF file in the existing Flash Player to upgrade users to the latest version of the player.

New ActionScript Class dialog

File | New | ActionScript Class

Use this dialog to specify the settings for an ActionScript class or interface to be created. You can also create a new package if you specify the package that doesn't yet exist.

Item	Description
Name	Specify the name of the class or interface. When the cursor is in this field, you can use the Up and Down arrow keys to browse the contents of the Template list .
Package	Specify the fully qualified name of the package in which the class or interface should be created. To do that, you can: – Select the package from the list. – Click ... (Shift+Enter) and select the package in the Choose Destination Package dialog that opens. – Type in the field. To enable package name completion, press Ctrl+Space . This field may be left blank. In this case, the class or interface will be created in the default root package (usually corresponds to the `src` folder). If you type the name of a package that doesn't yet exist (the name in this case is shown red), the corresponding package will be created. You can create more than one package at once. For example, if you type `myPackage.mySubpackage` and none of these packages currently exists, both these packages (`myPackage` and `mySubpackage`) will be created.
Template	Select the file template to be used. – Class. Select this template to create a class that doesn't extend another class or implement an interface. – Class with Supers. Select this template to create a class that extends another class and/or implements one or more interfaces. – Interface. Select this template to create an interface. To edit an existing file template or to create a new one, click ... (Shift+Enter). The File Templates dialog will open. If the selected template contains variables whose values are undefined, the Next button appears in the dialog instead of the Create button. In such a case, if you click Next, an additional dialog opens in which you can set the variable values. (This may happen if a custom template that contains custom variables is used.) See also, Predefined file template variables for ActionScript and Flex and An example of creating a custom file template for an MXML component.
Superclass	For the Class with Supers template: specify the class that the class you are creating should extend. To do that, you can: – Click ... (Shift+Enter) and select the class in the Choose Superclass dialog that opens. – Type in the field. To enable class name completion, press Ctrl+Space .
Interfaces	For the Class with Supers template: specify the interface or interfaces that the class you are creating should implement. To add an interface to the list, point to >> and click + (Alt+Insert). Select the interface in the Choose Super Interface dialog that opens. To remove unnecessary interfaces from the list, select them and click – (Alt+Delete).

New MXML Component dialog

File | New | MXML Component

Use this dialog to specify the settings for an MXML component to be created. You can also create a new package if you specify the package that doesn't yet exist.

Item	Description
Name	Specify the name of the component. This is the name of the `.mxml` file which will be created. When the cursor is in this field, you can use the Up and Down arrow keys to browse the contents of the Template list .
Package	Specify the fully qualified name of the package in which the component should be created. To do that, you can: – Select the package from the list. – Click … (Shift+Enter) and select the package in the Choose Destination Package dialog that opens. – Type in the field. To enable package name completion, press Ctrl+Space . This field may be left blank. In this case, the component will be created in the default root package (usually corresponds to the `src` folder). If you type the name of a package that doesn't yet exist (the name in this case is shown red), the corresponding package will be created. You can create more than one package at once. For example, if you type `myPackage.mySubpackage` and none of these packages currently exists, both these packages (`myPackage` and `mySubpackage`) will be created.
Template	Select the file template to be used. Initially, there is only one choice which is different depending on the Flex SDK version associated with the active build configuration (MXML 4 Component for Flex SDK 4 or MXML 3 Component for Flex SDK 3). – Class. Select this template to create a class that doesn't extend another class or implement an interface. – Class with Supers. Select this template to create a class that extends another class and/or implements one or more interfaces. – Interface. Select this template to create an interface. To edit an existing file template or to create a new one, click … (Shift+Enter). The File Templates dialog will open. If the selected template contains variables whose values are undefined, the Next button appears in the dialog instead of the Create button. In such a case, if you click Next, an additional dialog opens in which you can set the variable values. (This may happen if a custom template that contains custom variables is used.) See also, Predefined file template variables for ActionScript and Flex and An example of creating a custom file template for an MXML component.
Parent component	Specify a parent component for the component that you are creating. The parent component defines the root tag of the new MXML component, e.g. `<s:Application>` , `<s:Module>` , `<s:ComboBox>` . To specify the parent component, you can: – Click … (Shift+Enter) and select the component in the Choose Superclass dialog that opens. – Type in the field. To enable component name completion, press Ctrl+Space .

Package AIR Application Dialog

Build | Package AIR Application

Use this dialog to specify packaging options and to package your AIR applications (Desktop and Mobile) according to the specified options.

– The upper part of the dialog
– Packaging options

The upper part of the dialog

Use the available check boxes to select the build configurations for which you want to create the application packages.

Included in the list are the Desktop and Mobile-targeted build configurations whose output type is Application.

The build configurations are grouped by Flash modules.

Packaging options

Select the packaging options for Desktop and Mobile (Android and iOS) applications, and click Package.

Note that the availabilities of the options depend on the selected build configurations and their settings (e.g. on whether packaging for Android and iOS is enabled in the selected Mobile-targeted build configurations).

Item	Description
Desktop application package	Select one of the following: – installer (*.air). Select this option to create a digitally-signed installer file or a number of such files (one for each of the selected Desktop-targeted build configurations). – native installer. Select this option to create a native application installer or a number of native installers for the operating system that you are using. – captive runtime bundle. Select this option to create a captive runtime bundle or a number of bundles for the operating system that you are using. – unsigned package (*.airi). Select this option to create an unsigned AIR intermediate file or a number of such files.
Android package type (*.apk)	Select one of the following target types for the Android package or packages: – release. For the AIR runtime to be packaged, select the Captive runtime option. See the discussion of the captive runtime option. – debug over USB port. If necessary, change the port suggested by IntelliJ IDEA. – debug over network.
iOS package type (*.ipa)	Select one of the following target types for the iOS package or packages: – test without debugging. If necessary, select the Fast packaging option. If this option is selected, the ActionScript bytecode is interpreted and not translated to machine code. As a result, packaging is performed faster but code execution is slower. In technical terms, the `ipa-test-interpreter` target is used instead of `ipa-test`. See the discussion of the corresponding targets in Adobe AIR documentation. – debug over network. If necessary, select the Fast packaging option. (The `ipa-debug-interpreter` target will be used instead of `ipa-debug`.) – ad hoc distribution. – Apple App Store distribution.
Package	Click this button to package your applications.

Runtime-Loaded Modules dialog

Use this dialog to manage dependencies on runtime-loaded modules (RLMs). The RLMs are identified by their main classes.

To add a dependency, click + (`Alt+Insert`) and select the main class of the corresponding RLM in the Choose Main Class of Runtime-Loaded Module dialog that opens.

To optimize (reduce) the size of compiled RLM file, select the Optimize check box.

For more information, see the following table.

Item	Description
+ (`Alt+Insert`)	Use this icon or shortcut to add a dependency. Select the main class of the corresponding RLM in the Choose Main Class of Runtime-Loaded Module dialog that opens.
– (`Alt+Delete`)	Use this icon or shortcut to remove the selected RLMs from the list of dependencies.
Main Class	In this column, the fully qualified names of the main RLM classes are shown. To replace a class with a different one, click the corresponding field (table cell) and click ... that appears in the field (`Shift+Enter`). Then select the necessary class in the Choose Main Class of Runtime-Loaded Module dialog that opens.
Output File	In this column, paths to RLM SWF files to be generated are shown (readonly). All the paths are relative to the build configuration output folder.
Optimize	Select the check box for the corresponding RLM SWF file size to be optimized (reduced).

Diagram Reference

This section provides important information that is common for all types of diagrams used in IntelliJ IDEA. A diagram enables you to visually explore the dependencies, navigation rules, relationships etc. for the various types of applications.

In this section:

– Diagram Toolbar and Context Menu
– Diagram Preview
– General Techniques of Using Diagrams
– Class Diagram Toolbar and Context Menu
– Spring Dependencies Diagram

Diagram Toolbar and Context Menu

In this section:

– Toolbar

– Context menu

Toolbar

Item	Description
	Click this button to show primary key columns in diagram.
	Click this button to show columns in diagram.
	Click this button to increase the scale of the diagram. Alternatively, press `NumPad+`.
	Click this button to decrease the scale of the diagram. Alternatively, press `NumPad-`.
	Click this button to restore the actual size of the diagram.
	Click this button to make the contents fit into the current diagram size.
	Click this button to apply the current layout, selected on the context menu of the diagram.
	Click this button to save the current diagram in the specified location as `xml` file.
	Click this button to save the diagram in an image file with the specified name and path. The possible formats are: `jpeg`, `png`, `svg`, `svgz`, or `gif`.
	Click this button to print the diagram.
	Click this button to open the diagram preview in a separate frame, where you can configure the page layout, scale, and headings information.

Context menu

The table below contains commands that are not available from the toolbar.

Item	Description
New	Use this node to add new elements to a diagram.
Analyze	This node contains analysis commands, enabled in the current context.
Refactor	This node contains refactoring commands, enabled in the current context.
Jump to Source	Choose this command to open the selected diagram node element in the editor.
Find Usages	Choose this command to search for usages of the selected node element.
Layout	Select the desired diagram layout from the submenu.
Show Edge Labels	Check this command to show multiplicities in diagram.

Diagram Preview

Use the Structure view as a Preview, that allows you to get a "10,000-feet" look on a diagram. The shadow area represents the visible part of a diagram. As you zoom in or out, or change the shape of the IntelliJ IDEA windows, the size of the shadow area changes accordingly.

To enable the diagram preview

- Open the Structure tool window.

In the Preview pane, the following actions are available:

- Keeping the mouse button pressed, move the shadow area to obtain the desired view.
- Select one or more nodes in the diagram, and the corresponding nodes in the Preview are marked dark gray.

General Techniques of Using Diagrams

This section describes some general techniques applicable to the various types of diagrams (Hibernate and JPA ER diagrams, Seam navigation rules, Grails domain classes dependencies etc.)

In this section:

- Selecting elements in diagram .
- Managing diagram layout .
- Zooming in and out .
- Using the magnifier tool .
- Navigating to source code .
- Invoking refactoring commands .
- Finding usages of the selected node element .
- Drawing links between node elements .

To select elements in diagram

- To select an element, just click it in diagram.
- To select multiple adjacent elements, keep `Shift` pressed and click the desired elements, or just drag a *lasso* around the elements to be selected.
- To select multiple non-adjacent elements, keep `Ctrl+Shift` pressed and click the desired elements.
- To select a member of a node element, double-click the node element, and then use the arrow keys, or the mouse pointer.

To manage diagram layout

- Right-click the diagram background, and choose Layout command of the diagram context menu. Next, select the desired layout from the submenu.
- Use Drag-and-drop technique to lay out entities in diagram manually.
- Apply the current layout selected on the context menu of the diagram, by clicking .

To zoom in and out, do one of the following

- Use the and toolbar buttons.
- Keeping `Ctrl` key pressed, rotate your mouse wheel up or down.
- Press `NumPad+` or `NumPad-`.

> **Tip** As you zoom in or out, the size of the shadow area in the diagram preview changes accordingly.

To use the magnifier tool

- Keep the `Alt` key pressed, and hover your mouse pointer over the most interesting, or problematic areas of the diagram.

To jump from an element in diagram to the underlying source code

1. Select an element in diagram.
2. Do one of the following:
   - On the context menu of the diagram, choose Jump to Source
   - Press `F4`.
   - Double-click selected element.

   The source code of the corresponding source file opens in a separate tab in the editor.

To draw a link between nodes

- Click the source node, and drag a link to the target node. This technique slightly differs for the various types of diagrams. Refer to the corresponding procedures for details.

Class Diagram Toolbar and Context Menu

In this section:

– Toolbar

– Context Menu

Toolbar

Item	Description
	Click this button to show methods in the class nodes.
	Click this button to show fields in the class nodes.
	Click this button to show constructors in the class nodes.
	Click this button to show properties in the class nodes.
	Click this button to show inner classes in the class nodes.
	Click this button to reveal the combo box, and select visibility level of the elements to be displayed in diagram.
	Click this button to reveal the combo box, and select the desired scope of elements to be displayed in diagram, for example, project or non-project files. The elements out of the selected scope will be hidden.
	Click this button to enable creating extends or implements links between node elements. If this button is not pressed, links cannot be drawn.
	Click this button to show dependencies of the selected class or package.
	Click this button to increase the scale of the diagram, or press `NumPad+`.
	Click this button to decrease the scale of the diagram, or press `NumPad-`.
	Click this button to restore the actual size of the diagram.
	Click this button to make the contents fit into the current diagram size.
	Click this button to apply the current layout, selected on the context menu of the diagram, or press `F5`.
	Click this button to save the current diagram as a `*.uml` file.
	Click this button to save the diagram in an image file with the specified name and path. The possible formats are: `jpeg`, `png`, `svg`, `svgz`, or `gif`.
	Click this button to print the diagram.
	Click this button to open the diagram preview in a separate frame, where you can configure the page layout, scale, and headings information.

Context Menu

This section describes only those context menu commands that are not available from the toolbar.

Item	Shortcut	Description
Add class to diagram	`Space`	Choose this command to add existing class to the diagram background.
Collapse nodes	`C`	Choose this command to show the containing package of the selected node.
Expand nodes	`E`	Choose this command to show class diagram of the selected package.
New	`Alt+Insert`	Choose this command to create a new node element or member .
Refactor		Point to this node to select one of the refactoring commands available in this context.
Analyze		Point to this node to select one of the code analysis commands available in this context.

Spring Dependencies Diagram

The Spring dependencies diagram lets you view and analyze dependencies between your beans. It also lets you view and analyze a relationship between configuration files.

To access the Spring dependencies diagram: Diagrams | Show Diagram | Select Diagram Type in the context menu or `Ctrl+Shift+Alt+U` for a Spring configuration file selected in the Project tool window or opened in the editor.

Select Diagram Type

Item	Description
	Select this option to view beans that were defined in Spring configuration files. This type of diagram is most helpful when you need to view all autowired beans, for example, because it's hard to find another way to have them all displayed simultaneously.
	Select this option to view a relationship between configuration files. It might be useful when you trying to understand the structure of the setup instead of just the resulting beans in your application.

Note The toolbar options and context menu commands are common for all diagrams .

You can also use the diagram settings to change the toolbar options.

GUI Designer Reference

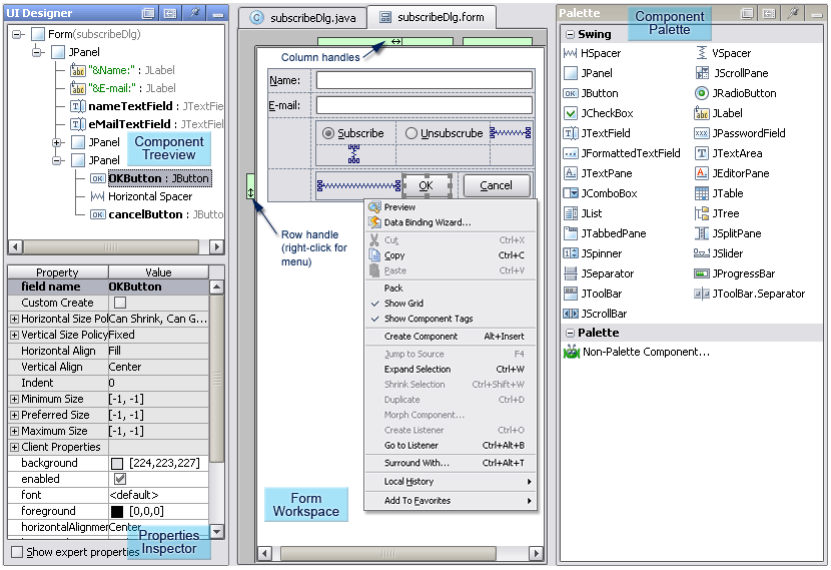
This part provides miscellaneous information related to the GUI Designer, including the tour of the UI, properties reference and dialog descriptions:

- Components of the GUI Designer
- Components Properties
- Components Treeview
- Data Binding Wizard
- Form Workspace
- Inspector
- Palette
- Add/Edit Palette Component

Components of the GUI Designer

The GUI Designer occupies the same space in the IntelliJ IDEA main window as the Editor. Each GUI form opens in a separate tab. (For information on how to create and open forms, see Building GUI Forms). The GUI Designer consists of the following main components:

- Components Treeview
- Properties Inspector
- Components Palette
- Form Workspace



Components Properties

The gray shaded section of the Inspector provides a set of properties that are proprietary to the GUI Designer and used by its code generation and other processes.

When a component is added to a form, it can be created as a component or as a container . In the latter case, such component acquires certain properties that are specific to the containers only. In the tables below the properties that pertain to the containers are specially noted.

In this section you will find descriptions of the following groups of properties:

- Code Binding Properties
- Component Sizing Properties
- Layout and Alignment Properties
- Other Properties

Properties are layout-specific. Some of the properties can be missing for certain layout managers.

Code Binding Properties

The properties covered in this section are related to binding of GUI forms and components to source code.

Property	Description
bind to class	This is a property of Forms only. It specifies the name of a class that contains the logic to make the form work. When this property is set to a valid class, we say the Form is bound to the class . If no target class exists yet, you can still type in the name of this *future* class. IntelliJ IDEA will offer a Quick Fix to create a class of the specified name for you whenever the property is focused. You can use the Quick Fix to create the class whenever you are ready. See sections Binding Form to Existing Class and Binding Form to a New Class .
field name	This is a property of components. It specifies the name of the field in the parent form's class to which the component is bound. For most components, a default field name is automatically entered, and a corresponding declaration is written to the Form's bound source file. You can change the default field name in the Inspector if you wish, and the source will be updated automatically. You can optionally change the field name in the source file, and the change will reflect in the Inspector when you return to the GUI Designer.
Custom Create	This is a property of components. If the option is checked, it means that you want to call a non-default constructor for the component, rather than have the GUI Designer generate a default constructor during the runtime build of the GUI. Code generation will ignore the component and assume you have written a constructor method. See Creating Form Initialization Code . If a non-default constructor does not yet exist, IntelliJ IDEA will show you a Quick Fix whenever the Custom Create property is focused in the Inspector. You can use this to create the constructor in the source file bound to the parent Form.

Component Sizing Properties

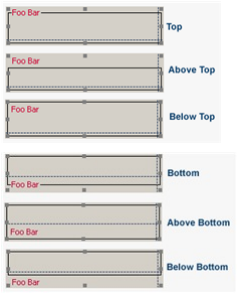
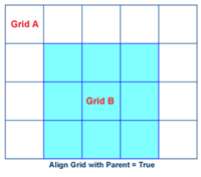
The properties described in this section affect how components are sized at design time and/or runtime.

Property	Subproperty	Description
Horizontal Size Policy Vertical Size Policy		These properties define how dimensions of a component are affected by resizing of its container along the horizontal axis and vertical axis respectively. This property applies to the GridLayoutManager (IntelliJ) and has the following subproperties:
	canShrink	The element size can be diminished (less than the preferred size) when the pane is resized.
	canGrow	The element size can be enlarged when the pane is resized.
	wantGrow	The element size is enlarged when the pane is resized. This flag takes precedence over canGrow .
		These options can be set simultaneously.
Same Size Horizontally		This is the property of a container that wraps the component. When the option is checked, all columns in the layout grid are always sized equally. Applicable only to the GridLayoutManager(IntelliJ).
Same Size Vertically		This is the property of a container that wraps the component. When the option is checked, all rows in the layout grid are always sized equally. Applicable only in the GridLayoutManager(IntelliJ).
Minimum Size Preferred Size Maximum Size		For the Swing layout managers, these properties are the same as in Java SDK. For the layout managers GridLayoutManager(IntelliJ) and FormLayout, these properties are different from those used in the Java SDK. To be more specific, they are not actual properties but a part of constraints with which a component is added to a container. Such feature enables you to set a size value only for 1 dimension. For instance, if you set Preferred Size values as `200; -1` - it means that the component height will be calculated dynamically and the width value will be used as the preferred size. In effect it is like the Java statement: `getPreferredSize().height();`

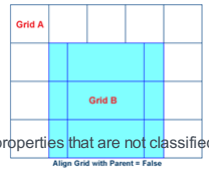
Layout and Alignment Properties

The properties described in this section control various aspects of component layout and/or alignment.

Property	Subproperty	Description
Layout Manager		This is a property of container type components only (e.g. JPanel, JScrollPane). The setting controls which layout manager the container uses. The setting affects both design-time and runtime. Find the list of supported layout managers in the section GUI Designer Options of the Settings dialog.
border		Defines how the component border and (optional) title will look. Applies to container type components only, and includes the following subproperties:
	type	Specifies the bevel characteristics of the element border. – None. No border. If *title* is specified, it will be hidden.

		– Empty. No border properties specified. – Bevel Lowered. Border bevel will make container look lowered. – Bevel Raised. Border bevel will make container look raised. – Etched. The container appears flat with an etched or *3D* border.
	title	Optionally specify a string to appear as the container's title at runtime. You can enter a string literal directly in the edit field, or click the ellipsis button to open a dialog where you can either hard-code a string value, or specify the identifier of a resource.
	title justification	Controls how the text of *title* is justified. – Default. The justification is determined at compile-time. – Left. Force left-justified text. – Center. Force centered text. – Right. Force right-justified text. – Leading. For use with locales requiring leading justification. – Trailing. For locales requiring trailing justification.
	title position	Controls where the title is positioned with respect to the container border. Horizontal position of the text is controlled by title justification setting. Default, Above Top, Top, Below Top, Above Bottom, Bottom, Below Bottom, as shown on the following images: 
	title font	Controls the font used for displaying the text of the title.
	title color	Controls the color of the font used for displaying the text of the title. The ellipsis button launches a color picker dialog, where you can choose a color from one of several palettes (AWT, Swing, System), or specify an RGB or HSB color value, or select a color from a graphical set of color Swatches.
margins		This property of JPanel controls the amount of spacing between the outer border of the container, and its contents. Applies to FormLayout and GridLayoutManager(IntelliJ).
	Top Right Bottom Left	Each attribute controls the spacing at the respective edge of the pane. The value of each attribute is an integer which specifies the number of pixels in the respective spacing. Zero means no space.
Horizontal Gap Vertical Gap		This is a property of JPanel only and has effect only when the pane uses a grid type layout manager such as the default GridLayoutManager. The property defines the pixel dimension of a space inset between the edge of a grid cell in the pane, and the edge of a contained component (a JRadiobutton, for example). The default value is -1, which indicates the default spacing. You can enter zero or any positive integer value and see the result at design time. For Vertical Gap to have any effect, the layout grid should have at least 2 rows.
Horizontal Align		This property determines the relative horizontal position of a component within its container. Select a value from the drop-down list: – Left. The left-hand edge of the component snaps to the left border of its container. – Center. The component is centered horizontally within its container. – Right. The right-hand edge of the component snaps to the right border of its container. – Fill. The component fills its container's horizontal space entirely.
Vertical Align		This property determines the relative vertical position of a component within its container. Select a value from the drop-down list: – Top. The top edge of the component snaps to the top border of its container. – Center. The component is centered vertically within its container. – Bottom. The bottom edge of the component snaps to the bottom border of its container. – Fill. The component fills its container's vertical space entirely.
Indent		This property is only applicable for GridLayoutManager. Valid values are zero or any positive integer. The selected component is shifted to the right by the specified number of pixels times ten. For example, if you enter 12, the indent will be 120 pixels (12 * 10).
Align Grid with Parent		This is a property of panes and applicable with grid type layout managers. When checked, it means that grid columns and rows in a child (nested) container always align with the rows and columns of the parent container. If not checked, the grid columns and rows of a child container may be aligned independently: 

Other Properties

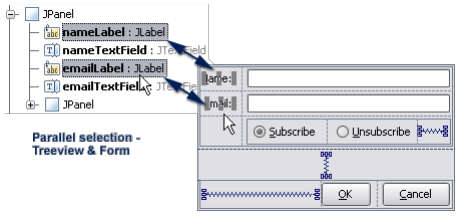


This section describes other GUI Designer properties that are not classified any other way.

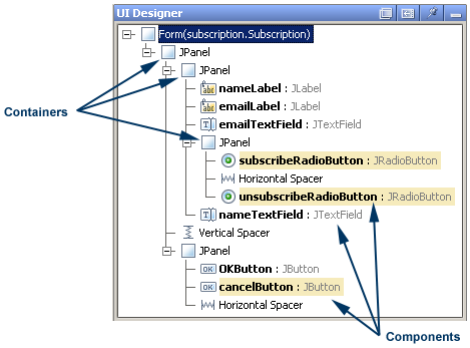
Item	Description
Client Properties	This is a property with configurable sub-properties. If you develop your own components, you can configure a Custom Property in the GUI Designer to support it. Refer to the section Customizing Client Properties .

Components Treeview

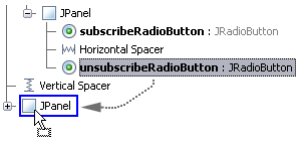
This treeview displays the components contained in the design form and enables you to navigate to and select one or more components. Selection of one or more components here is reflected in parallel on the design form and vice versa as shown in the following figure:



The treeview hierarchy represents containment. Expandable nodes represent some type of container. Sub-nodes of containers represent UI components (including nested containers). The root node represents the Form, which is, in effect, the top-level container for the GUI you are building with the GUI Designer. When you create a new Form, a JPanel component is automatically added to the Form Workspace and it appears as a child of the Form in the Component Treeview. This JPanel is the top of the UI component hierarchy (in the Java sense) for the current Form. All other Swing or other UI components are contained by it, as the next figure illustrates:



It is possible to move components from one container to another using drag-and-drop operation in the Component Treeview:



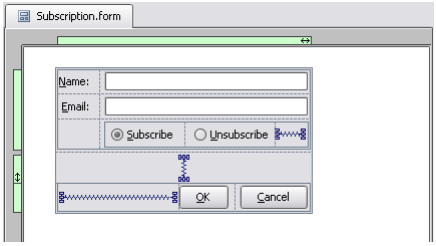
Data Binding Wizard

Use this dialog box to generate getData and setData methods for the fields in a UI class that are bound to components in a GUI form.

Item	Description
Page 1	
Create new bean	Click this radio button to bind components to data in a new bean class. If this option is selected, specify the name of the new bean class, and the package where it will be created.
Bind to existing bean	Click this radio button to bind components to data in an existing bean class. If this option is selected, specify the name of the desired bean class.
Page 2	
Form Field	This column displays the list of components of the GUI Form that can be bound to data.
Bean Property	Use this column to specify the name of the bean property that will be created in the specified bean class.
getData()	If this option is checked, the method getData() is generated in the bound class of the GUI form.
setData()	If this option is checked, the method setData() is generated in the bound class of the GUI form.
isModified()	If this option is checked, the method isModified() is generated in the bound class of the GUI form.

Form Workspace

The Form Workspace occupies the center part of the frame (assuming the default tool window layout and visibility). The background is white by default. When you create a new Form, a JPanel component is added to the workspace which appears as a gray rectangle. You can place components from the Component Palette into this container by first clicking on the component in the Palette, and then clicking within the pane in the Form Workspace. The form workspace has a context menu that provides access to the Clipboard, layout actions and more.



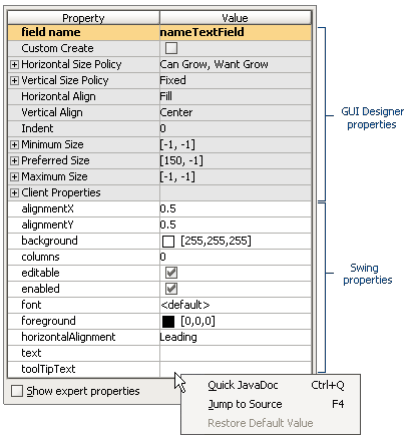
Item	Description
Preview	Show the form as it should look at runtime. See Previewing Forms .
Data Binding Wizard	Generate `getData()` , `isModified()` and `isModified()` methods for the fields bound to data. The corresponding class should already exist. See Generating Accessor Methods for Fields Bound to Data
Cut, Copy, Paste	Perform usual Clipboard operations.
Pack	Choose this command to compress the current form to its minimal size, defined by the layout manager. This command is only available for the top level container in a form.
Show Grid	If this option is checked, the form displays grid lines.
Show Component Tags	This option toggles display of the name of a field associated with the selected component. This feature is available for the components that exceed certain pre-defined dimensions.
Create Component	Choose this command to display the list of available components and insert the selected one in the current location.
Jump to Source	Open the bound class in the editor, and place the caret to the field associated with the selected component. For the whole form, the caret rests at the class declaration.
Expand/Shrink Selection	Select successively increasing sets of components from the current one to its container. Compare to selecting text in the editor .
Duplicate	Clone the selected component.
Morph Component	Create a component of a different type with the same properties. See Morphing Components .
Create Listener	Create listener for the selected component .
Go To Listener	Navigate to the source code of the selected listener.
Surround With	Display the list of available containers, and place in one or more selected components into the container of your choice. See section Wrapping/Unwrapping Components
Flatten	Unwrap components from a container. See section Wrapping/Unwrapping Components .
Local history	Access the commands of the local version control .
Add to Favorites	Add selected component to favorites .

Inspector

The Property Inspector window shows properties for the component currently selected in the form workspace, or the form itself if no components exist or none are selected.

In this section you will find information about the groups of properties, context menu commands, and types of editors.

The Inspector has two groups of properties, as shown in the following figure:



Item	Description
Upper group	The shaded properties at the top of the Inspector are proprietary to IntelliJ IDEA; they are mainly used to control the layout constraints of the components for the given layout. These properties are layout-specific and depend on the layout of the container where the component is placed.
Lower group	This group is not shaded and contains properties of the selected Swing component. There are two levels of properties: *Basic* and *Expert*. The Expert level can be toggled on and off using the Show Expert Properties check box in the bottom line of the Inspector. Refer to Sun documentation for the Swing libraries .

The context menu of each property provides the following commands:

Item	Keyboard Shortcut	Description
Quick Javadoc	Ctrl+Q	Opens related API documentation for the selected property, provided that the necessary paths are added to the API docs in the Project Settings.
Jump to Source	F4	Opens in the editor the source code of the class that contains the selected property.
Restore Default Value		This command is enabled for the modified properties only.

Several types of property editors appear in the Value column of the inspector:

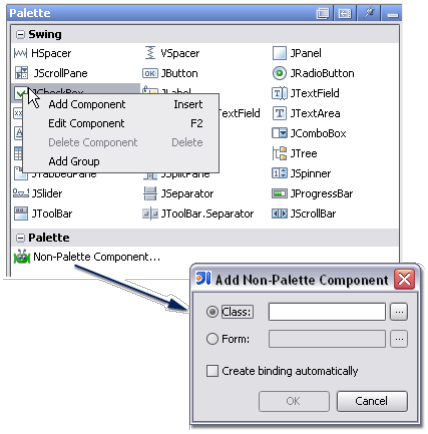
- Text field: Type a value.
- Pick list: Pick a value from a drop-down list of valid choices.
- Check box: Set value for Boolean type properties.
- Dialog: Presents an ellipsis button which opens a dialog box.

Palette

The Component Palette is a tool window which appears by default at the right side of the frame next to the Form Workspace. It contains UI components which you can visually place on a Form. The default Group of the Palette contains a set of Swing user interface components that you can arrange on forms as needed. The Swing group also has horizontal and vertical Spacers that you can place on the form to define space between components. (These can behave differently depending on the setting in the Layout Manager property of the container into which a spacer is placed.) You can customize the Component Palette to contain additional groups, and your own and/or third-party GUI components (see Customizing the Palette).

The context menu of the Palette tool window provides functions for managing components and groupings. Two groups are present by default:

- Swing: contains components from the Swing component library.
- Palette: contains a single component labeled *Non-Palette component*. When you select this *component* and add it to a Form, a dialog appears in which you can select any component class accessible to your project, or any other existing Form. This is useful in cases where you want to use a component without adding it to the Component Palette.

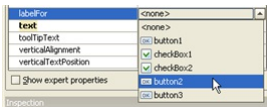


If you have your own custom components, or if you reuse components from third-party libraries, you can add them to the Component Palette via the context menu, which also enables you to add component groups. For more information, see Customizing the Component Palette .

Add/Edit Palette Component

GUI Designer Palette | context menu | Add / Edit Component

Use this dialog to create a new component in the Palette , or change an existing one.

Item	Description
Class	Click this radio-button to add a component from a class library. You can enter a fully qualified class name in the text field, or click the ellipsis button, and select the desired class from the libraries, or project. Note that code completion is available in the text field.
Form	Click this radio-button to add an existing GUI form. You can enter a fully qualified form name in the text field, or click the ellipsis button, and select the desired form from the libraries, or project.
Icon	Specify the fully-qualified name of the icon file, or click the ellipsis button, and select the desired icon file from the libraries, or project.
Group	Select the target group where the new component will be added. Tip This field is available for the Add Component dialog only.
Horizontal / Vertical Size Policy	The sizing policies define the behavior of the component, when its parent container is being resized.
Can shrink	If this option is checked, the size of the component can reduce, when the container is resized.
Can grow	If this option is checked, the size of the component can increase, when the container is resized.
Want grow	If this option is checked, the size of the component can increase, when the container is resized. This option has a higher priority, when competing with the cells of the other components.
Is container	If this option is checked, the component can accommodate nested components, and acquires some properties that pertain to the containers.
Create binding automatically	If this option is checked, the field for the component is automatically added to the bound class .
Can have attached label	If this option is checked, the component of this type appears in the `labelFor` field of a `JLabel` component in the Inspector . It is important to note that Can have attached label option affects all components of this type on a GUI form. For example, if you check this option for `JButton` and `JCheckBox` Palette components, all instances of these components (already existing and newly added) appear in the `labelFor` field: 

Icons Reference

– File Types Recognized by IntelliJ IDEA
– Symbols

File Types Recognized by IntelliJ IDEA

IntelliJ IDEA recognizes numerous file types. Each file type is denoted with a special icon. Custom files types are also allowed. Each file type is associated with one or more extensions that match a certain pattern.

The file types and their extensions are configurable in the File Types dialog.

Note "Recognized" does not mean "supplied with extensive support". For example, `php` files are recognized in the Community Edition and marked with the corresponding icon, although the edition does not provide PHP development support.

Note The file recognized types depend on the installed plugins.

The default types include:

File Type	Icon	Recognized in
ActionScript files		Ultimate Edition
Active Server Pages files		Ultimate Edition
Android files		Ultimate Edition: all Android-related file types; Community Edition: Android IDL files, Android renderscript files
Apache Config files		Ultimate Edition
Archive files		Ultimate Edition, Community Edition
AspectJ files		Ultimate Edition, Community Edition
C# files		Ultimate Edition, Community Edition
C/C++ files		Ultimate Edition, Community Edition
Command Shell files		Ultimate Edition
CSS files		Ultimate Edition
CoffeeScript files		Ultimate Edition
Cucumber feature files		Ultimate Edition
ColdFusion files		Ultimate Edition
Eclipse project files		Ultimate Edition, Community Edition
Dart files		Ultimate Edition, Community Edition
Diagram files		Ultimate Edition
Drools files		Ultimate Edition
Erlang files		Ultimate Edition, Community Edition
EJB QL files		Ultimate Edition, Community Edition
Files marked as plain text		Ultimate Edition, Community Edition
Files opened in associated applications		Ultimate Edition, Community Edition
FreeMarker template files		Ultimate Edition
Gant scripts		Ultimate Edition
Gradle scripts		Ultimate Edition
Groovy files		Ultimate Edition, Community Edition
Groovy Server Pages		Ultimate Edition
GUI Form		Ultimate Edition, Community Edition
Handlebars files		Ultimate Edition
HAML files		Ultimate Edition
HTML files		Ultimate Edition, Community Edition
IntelliJ IDEA project, module or workspace files.		Ultimate Edition, Community Edition
IDL files		Ultimate Edition, Community Edition
Image files		Ultimate Edition, Community Edition
Java class files		Ultimate Edition, Community Edition
Java source files		Ultimate Edition, Community Edition
JavaFX files		Ultimate Edition, Community Edition
JavaScript files		Ultimate Edition
JavaScript test files		Ultimate Edition
JavaScript files that can be executed on Node.js		Ultimate Edition
Jade files (refer to the section Using Pug (Jade) Template Engine).		Ultimate Edition
JSF files		Ultimate Edition, Community Edition
JSHint configuration files		Ultimate Edition

JSON files		Ultimate Edition
JSTestDriver Config files		Ultimate Edition
Java Server Pages files		Ultimate Edition, Community Edition
JSPx files		Ultimate Edition, Community Edition
Kotlin files		Ultimate Edition, Community Edition
Kotlin classes		Ultimate Edition, Community Edition
Kotlin interfaces		Ultimate Edition, Community Edition
Kotlin enums		Ultimate Edition, Community Edition
Kotlin objects		Ultimate Edition, Community Edition
Less files		Ultimate Edition
Patch files		Ultimate Edition, Community Edition
Perl files		Ultimate Edition, Community Edition
PHP files		Ultimate Edition, Community Edition
Properties files		Ultimate Edition, Community Edition
Resource bundles		Ultimate Edition, Community Edition
XML-based properties files		Ultimate Edition, Community Edition
Pug files (refer to the section Using Pug (Jade) Template Engine).		Ultimate Edition
Regular expressions		Ultimate Edition, Community Edition
RELAX NG Compact Syntax		Ultimate Edition, Community Edition
Sass files		Ultimate Edition
SCSS files		Ultimate Edition
Scala files		Ultimate Edition
Smarty, Smarty config files		Ultimate Edition
SQL files		Ultimate Edition, Community Edition
Stylus files		Ultimate Edition
Drools Expert files		Ultimate Edition
Text files		Ultimate Edition, Community Edition
TypeScript files		Ultimate Edition
Velocity template files		Ultimate Edition
XHTML files		Ultimate Edition, Community Edition
XML DTD files		Ultimate Edition, Community Edition
XML files		Ultimate Edition, Community Edition
YAML files		Ultimate Edition

Symbols

In this section:

– Common

– Data Sources

Common

Icon	Description
	Class
	Abstract Java class
	Annotation
	Enumeration
	Exception
	Final Java class
	Interface
	Java class that contains declaration of the `main()` method.
	Test case
	Java class located out of the source root. Refer to the section Configuring Content Roots for details.
	Java class excluded from compilation .
	Method
	Abstract method
	Field
	Variable
	Property
	Parameter
	Element
	Directory
	Module
	Group of modules
	Package
	Source root
	Test root
	Excluded root
	Resources
	Test resources
	Generated source roots
	Generated test source roots
Visibility modifiers	
	Read-only class, e.g. from a jar of an external library.
	private
	protected
	package protected
	static
	public

Data Sources

Icon	Description
	DB data source. Also, DBMS-specific icons are used: DB2 Derby H2 HSQLDB MySQL Oracle PostgreSQL SQL Server SQLite

	Sybase
	DB data source with the read-only status, e.g. for DB2.
	DDL data source
	Schema
	Table
	View
	Column
	A `NOT NULL` column
	Column with a primary key
	Column with a foreign key
	Column with an index
	Primary key
	Foreign key
	Index
	Stored procedure or function

Regular Expression Syntax Reference

This section provides a brief summary of regexp syntax that can be helpful for creating search and issue navigation patterns.

In this section:

– RegEx Syntax Reference

– Tips and Tricks

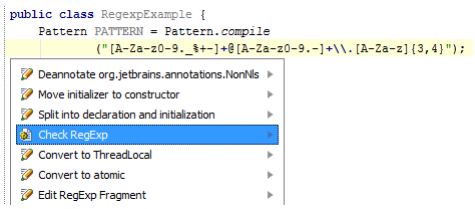
RegEx Syntax Reference

Character	Description
`\`	Marks the next character as either a special character or a literal. For example: – *n* matches the character *n*. "\n" matches a newline character. – The sequence `\\` matches `\` and `\(` matches `(`.
`^`	Matches the beginning of input.
`$`	Matches the end of input.
`*`	Matches the preceding character zero or more times. For example, "zo*" matches either *z* or *zoo*.
`+`	Matches the preceding character one or more times. For example, "zo+" matches *zoo* but not *z*.
`?`	Matches the preceding character zero or one time. For example, `a?ve?` matches the `ve` in `never`.
`.`	Matches any single character except a newline character.
`(` subexpression `)`	Matches *subexpression* and remembers the match. If a part of a regular expression is enclosed in parentheses, that part of the regular expression is grouped together. Thus a regex operator can be applied to the entire group. – If you need to use the matched substring within the same regular expression, you can retrieve it using the backreference (`\num`, where `num = 1..n`). – If you need to refer the matched substring somewhere outside the current regular expression (for example, in another regular expression as a replacement string), you can retrieve it using the dollar sign (`$num`, where `num = 1..n`). – If you need to include the parentheses characters into a *subexpression*, use `\(` or `\)`.
x `|` y	Matches either *x* or *y*. For example, `z|wood` matches *z* or *wood*. `(z|w)oo` matches *zoo* or *wood*.
{ `n` }	*n* is a nonnegative integer. Matches exactly *n* times. For example, `o{2}` does not match the *o* in *Bob*, but matches the first two o's in *foooood*.
{ `n` ,}	*n* is a nonnegative integer. Matches at least *n* times. For example, `o{2,}` does not match the *o* in *Bob* and matches all the o's in "foooood." `o{1,}` is equivalent to `o+`. `o{0,}` is equivalent to `o*`.
`{` *n* `,` *m* `}`	*m* and *n* are nonnegative integers. Matches at least *n* and at most *m* times. For example, `o{1,3}` matches the first three o's in "fooooood." `o{0,1}` is equivalent to `o?`.
`[` *xyz* `]`	A character set. Matches any one of the enclosed characters. For example, `[abc]` matches the *a* in *plain*.
`[^` *xyz* `]`	A negative character set. Matches any character not enclosed. For example, `[^abc]` matches the *p* in *plain*.
`[` *a-z* `]`	A range of characters. Matches any character in the specified range. For example, "[a-z]" matches any lowercase alphabetic character in the range *a* through *z*.
`[^` *m-z* `]`	A negative range characters. Matches any character not in the specified range. For example, `[m-z]` matches any character not in the range *m* through *z*.
`\b`	Matches a word boundary, that is, the position between a word and a space. For example, `er\b` matches the *er* in *never* but not the *er* in *verb*.
`\B`	Matches a non-word boundary. `ea*r\B` matches the *ear* in *never early*.
`\d`	Matches a digit character. Equivalent to `[0-9]`.
`\D`	Matches a non-digit character. Equivalent to `[^0-9]`.
`\f`	Matches a form-feed character.
`\n`	Matches a newline character.
`\r`	Matches a carriage return character.
`\s`	Matches any white space including space, tab, form-feed, etc. Equivalent to `[ \f\n\r\t\v]`.
`\S`	Matches any nonwhite space character. Equivalent to `[^ \f\n\r\t\v]`.
`\t`	Matches a tab character.
`\v`	Matches a vertical tab character.
`\w`	Matches any word character including underscore. Equivalent to `[A-Za-z0-9_]`.
`\W`	Matches any non-word character. Equivalent to `[^A-Za-z0-9_]`.
`\` num	Matches *num*, where *num* is a positive integer, denoting a reference back to remembered matches. For example, `(.)\1` matches two consecutive identical characters.
`\` *n*	Matches *n*, where *n* is an octal escape value. Octal escape values should be 1, 2, or 3 digits long. For example, `\11` and `\011` both match a tab character. `\0011` is the equivalent of `\001` & *1*. Octal escape values should not exceed 256. If they do, only the first two digits comprise the expression. Allows ASCII codes to be used in regular expressions.
`\x` *n*	Matches *n*, where *n* is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long.

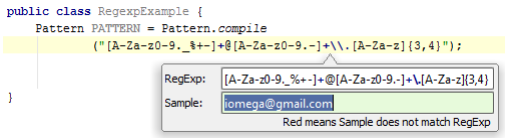
	For example, `\x41` matches `A`. `\x041` is equivalent to `\x04` & `1`. Allows ASCII codes to be used in regular expressions.
`\\$`	Escapes `$`.
`\l`	Changes the case of the next character to the lower case.
`\u`	Changes the case of the next character to the upper case.
`\L`	Changes the case of all the subsequent characters up to `\E` to the lower case.
`\U`	Changes the case of all the subsequent characters up to `\E` to the upper case.

Tips and Tricks

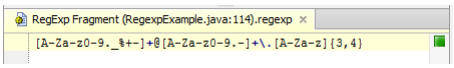
IntelliJ IDEA provides intention actions to check validity of the regular expressions, and edit regular expressions in a scratchpad. Place the caret at a regular expression, and press `Alt+Enter`. The suggestion list of intention actions, available in this context, appears:



- Choose Check RegExp, and press `Enter`. The dialog box that pops up, shows the current regular expression in the upper pane. In the lower pane, type the string to which this expression should match. If the regular expression matches the entered string, the background becomes green. If the regular expression doesn't match, then the background is red.



- Choose Edit RegExp Fragment, and press `Enter`. The regular expression opens for editing in a separate tab in the editor. However, this is but a scratchpad, and no file is physically created:



  As you type in the scratchpad, all changes are synchronized with the original regular expression. Press `Escape` to close the editor tab.

Scope Language Syntax Reference

The scopes language is used in specifying project scopes involved in various kinds of analysis.

In this section:

– Sets of Classes
– Sets of Files
– Modifiers
– Logical Operators
– Defining Scopes
– Examples

Sets of Classes

– Single class is defined by a class name, i.e. `com.intellij.openapi.MyClass`
– Set of all classes in a package, not recursing into subpackages, is defined by an asterisk after dot, for example: `com.intellij.openapi.*`
– Set of all classes in a package including contents of subpackages, is defined by an asterisk after double dot, for example `com.intellij.openapi..*`

Sets of Files

– Single file is defined by a file name, i.e. `MyDir/MyFile.txt`
– Set of all files in a directory, not recursing into subdirectories, is defined by an asterisk after slash, for example: `file:src/main/myDir/*`
– Set of all files in a directory including contents of subdirectories, is defined by an asterisk after double slash, for example `file:src/main/myDir//*`

Modifiers

Location modifiers

help you specify whether the desired set is located in the source files, library classes or test code in the form of location modifiers `src:`, `lib:`, `file:`, or `test:`.
For example, the following scope

```
src:com.intellij.openapi.*
```

implies all classes under the source root in the `com.intellij.openapi` package, excluding subpackages.

The default location is the module root.

Module modifiers

help you narrow down the scope by specifying the name of the related module in one of the following ways:

```
src[module name]:<E>
lib[module name]:<E>
test[module name]:<E>
```

For example, the following scope

```
src[MyModule]:com.intellij.openapi.*
```

implies all classes under the source folders related to the module `MyModule` in the package `com.intellij.openapi`, excluding subpackages.

Group modifier

help you narrow down the scope by specifying the name of the related module group (several modules can be joined into a group in the Project Structure dialog).
The group modifier has the following format:

```
[group:<group name>]
```

For example, the following scope

```
file[group:mygroup]:*//*
```

denotes a scope of all files in the group of modules with the specified name.

Logical Operators

The scope language allows you to use common logical operators:

```
&& for AND
|| for OR
! for NOT
```

Besides that, the parentheses can be used to join the logical operators into groups. For example, the following scope

```
(<a>||<b>)&&<c>
```

implies either <a> and <c>, or <b> and <c>.

Another example

```
file[*web*]:src/main/java//*
```

denotes a scope of all modules whose name contains `web`, and all the files recursively in the directory `src/main/java`

Defining Scopes

Scopes are defined in the Scopes dialog box in the following ways:

- Manually
- With the pointing device

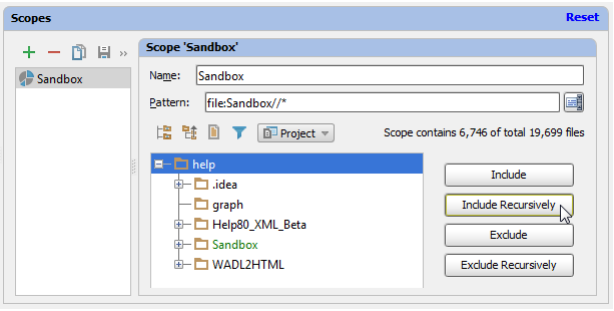
Manually

Specify file masks in the Pattern text box, or click and type the pattern in the editor.

Using the Mouse Pointer

Select files and folders in the project tree view and click the buttons Include, Include Recursively, Exclude, and Exclude Recursively. For information about the controls, refer to Scope page description.

Based on the inclusion/exclusion of file and directories, IntelliJ IDEA creates an expression and displays it in the Pattern field.



Examples

- `file[MyMod]:src/main/java/com/example/my_package//*` - include in a project all the files from module "MyMod", located in the specified directory and all subdirectories.
- `src[MyMod]:com.example.my_package..*` - recursively include all classes in a package in the source directories of the module.
- `lib:com.company..*||com.company..*` - recursively include all classes in a package from both project and libraries.
- `test:com.company.*` - include all test classes in a package, but not in subpackages.
- `[MyMod]:com.company.util.*` - include all classes and test classes in the package of the specified module.
- `file:*.js||file:*.coffee` - include all JavaScript and CoffeeScript files.
- `file:*js&&!file:*.min.*` - include all JavaScript files except those that were generated through minification, which is indicated by the `min` extension.

Index of Menu Items

IntelliJ IDEA menu structure doesn't align with IntelliJ IDEA help structure. This page lists IntelliJ IDEA menu items, linked to the corresponding help topics.

Tip This table corresponds to the Windows platform.

In this section:

– File
– Edit
– View
– Navigate
– Code
– Analyze
– Refactor
– Build
– Run
– Tools
– VCS
– Window
– Help

File

Menu item	Keyboard shortcut	Description
New		
		Project — Use this command to create a new project .
		Project from Existing Source — Use this command to create a new project from existing source .
		Project from Version Control — Use this command to check out a project from a version control system.
		Module — Use this command to add a new module to an existing project.
		Module from Existing Sources — Use this command to create a new module from existing sources.
		<File type> — Use this command to create a new element in a project .
Open		Use this command to open the specified directory, or an existing IntelliJ IDEA project. A directory that contains a project is marked with icon. Refer to the section Opening, Reopening, and Closing Projects . This command is duplicated with icon on the main toolbar.
Open URL		Use this command to open source code of a web page in the editor.
Open Recent		Use this command to open one of the recent projects. Refer to the reopening projects procedure.
Close Project		Use this command to close the current project . Refer to the sections Opening Multiple Projects and Opening, Reopening and Closing Projects for details.
Settings (on Windows/*NIX)/IntelliJ IDEA Preferences (on OS X)	Ctrl+Alt+S	Use this command to change the project and IDE configurations in the Settings/Preferences dialog . See also the section Configuring Project and IDE Settings . This command is duplicated with icon on the main toolbar. This command in available on Windows/Linux. On Mac OS it appears on the IntelliJ IDEA menu and has the name Preferences.
Project Structure	Ctrl+Shift+Alt+S	Use this command to configure SDK, libraries, content roots etc.
Other Settings		Point to this node to reveal the submenu of the default configuration commands (Default Settings, Configure Plugins, Default Project Structure)
Import Settings...		Choose this command to import settings from an archive .
Export Settings...		Choose this command to export settings to an archive .
Export to Eclipse		Choose this command to invoke Export to Eclipse Dialog . As a result, two Eclipse files are added to the exported project: `.classpath` and `.project` . If in Export to Eclipse Dialog the check box Switch selected module to Eclipse-compatible format is selected, the file `<module name>.eml` is created. If in Export to Eclipse Dialog the check box Export non-module libraries is selected, the file `<module name>.userlibraries` is created. Refer to the section Exporting an IntelliJ IDEA Project to Eclipse for details.
Settings Repository...		Choose this command to invoke the Settings Repository dialog.
Save All	Ctrl+S	Choose this command to save all changes , when editing is over. This command is duplicated with icon on the main toolbar.
Synchronize	Ctrl+Alt+Y	Choose this command to check the IntelliJ IDEA caches and bring them up-to-date by keeping in sync with external changes. This command is duplicated with icon on the main toolbar.
Invalidate Caches/Restart...		Choose this command to clean the system cache .

Export to HTML...	Use this command results to save selected files in HTML format .
Print...	Choose this command to print selected file on the default printer. Refer to the Print dialog description.
Add to Favorites	Use this command to add the selected files to the list of Favorites . Click the right arrow to select the list of favorites you want to be modified. Refer to the description of the Favorites tool window .
File Encoding	Use this command to change encoding of an individual file . See also the section Encoding .
Line Separators	Use this command to select the desired line separator style. Refer to the section Configuring Line Separators .
Make File Read-Only / Make File Writable	Use these toggle commands to change read-only status of a file selected in the Project tool window, or currently active in the editor. If a file is made read-only, it is marked with , and doesn't allow editing. You can also toggle read-only attribute of a file in the Status bar .
Power Save Mode	Use this mode if you are working with a laptop. If Power-Save mode is on, then the background processes are turned off, to minimize the power consumption. You can also turn this mode on or off by clicking in the Status bar .
Exit	Choose this command to quit IntelliJ IDEA. This command in available on Windows/Linux. On Mac OS it appears on the IntelliJ IDEA menu and has the name Quit IntelliJ IDEA.

Edit

Menu item	Keyboard shortcut	Description
Undo <action>	Ctrl+Z	Use this command to roll actions back . This command is duplicated with icon on the main toolbar.
Redo <action>	Ctrl+Shift+Z	Use this command to repeat the last actions . This command is duplicated with icon on the main toolbar.
Cut	Ctrl+X	Choose this command to take the selected characters to the clipboard and delete them. Refer to the section Cutting, Copying, and Pasting . This command is duplicated with icon on the main toolbar.
Copy	Ctrl+C	Choose this command to take the selected characters to the clipboard. Refer to the section Cutting, Copying, and Pasting . This command is duplicated with icon on the main toolbar.
Copy Path	Ctrl+Shift+C	Choose this command to take the path to the selected symbol to the clipboard. Refer to the section Cutting, Copying, and Pasting .
Copy as Plain Text	Ctrl+Shift+C	Choose this command to take the selected fragment to the clipboard without formatting. Refer to the section Settings/Preferences | Editor | General .
Copy Reference	Ctrl+Shift+Alt+C	Choose this command to take a reference to a symbol to the clipboard. Refer to the section Cutting, Copying, and Pasting .
Paste	Ctrl+V	Choose this command to place the latest entry from the Clipboard at the insertion point. Refer to the section Cutting, Copying, and Pasting . This command is duplicated with icon on the main toolbar.
Paste from History...	Ctrl+Shift+V	Choose this command to place at the insertion point the selected entry from the Clipboard. Refer to the section Cutting, Copying, and Pasting .
Paste Simple	Ctrl+Shift+Alt+V	Choose this command to place the last entry from the Clipboard at the insertion point as plain text. Refer to the section Cutting, Copying, and Pasting .
Delete	Delete	Choose this command to delete the selected files, or folder from the project tool window, or selected fragment of text from the active editor.
Find		Point to this node to reveal the sub-menu of search commands:

Find/Replace	Ctrl+F / Ctrl+R	Find or replace text in a current file . These commands are duplicated by and icons on the main toolbar.
Find Next/Find Previous	F3 / Shift+F3	Use these commands to navigate through the search results in a file. See Finding and Replacing Text in File .
Find Word at Caret	Ctrl+F3	Use this command to jump to the next occurrence of the word where the caret rests. See Finding and Replacing Text in File .
Select All Occurrences	Ctrl+Shift+Alt+J	Use this command to find and select all the occurrences of an item.
Add Selection to Next Occurrence	Alt+J	use this command to select the next occurrence of an item.
Unselect Occurrence	Shift+Alt+J	Use this command to remove selection from the last selected occurrence of an item.
Find in Path/Replace in Path	Ctrl+Shift+F / Ctrl+Shift+R	Use these commands to search for, and replace a text fragment in a whole project. Refer to the section Finding and Replacing text in Project .
Search/Replace Structurally		Use these commands to perform structural search or replace. Refer to the section Structural Search and Replace for details.
Find Usages	Alt+F7	Use this command to search for the usages of a symbol across an entire project. Refer to the section Finding Usages in Project .

Find Usages Settings	Ctrl+Shift+Alt+F7	Use this command to search for the usages of a symbol across an entire project, after setting the desired search options. Refer to the section Finding Usages in Project .
Show Usages	Ctrl+Alt+F7	Use this command to bring up a list of the usages of a symbol across the whole project. Refer to the section Viewing Usages of a Symbol .
Find Usages in a File	Ctrl+F7	Refer to the section Finding Usages in the Current File .
Highlight Usages in a File	Ctrl+Shift+F7	Use this command to visualize usage of a symbol in the current file. Refer to the section Highlighting Usages .
Recent Find Usages	Ctrl+E	Choose this command to view the recent search results. Refer to the section Viewing Recent Find Usages .
Find by XPath	Ctrl+Alt+X, F	Use this command to o find occurrences of certain XPath expressions in all XML files in a specific scope.

Macros		Point to this node to reveal the sub-menu of the macros-related commands. Refer to the section Using Macros in the Editor .
Column Selection Mode	Shift+Alt+Insert	Use this command to toggle between column selection and line selection modes. Refer to the section Selecting Text in the Editor .
Select All	Ctrl+A	Choose this command to select all contents of the current file. Refer to the section Selecting Text in the Editor .
Extend Selection	Ctrl+W	Choose this command to select the current word. Use this command successively to extend selection. Refer to the section Selecting Text in the Editor .
Shrink Selection	Ctrl+Shift+W	Choose this command to unselect the currently selected word. Use this command successively to shrink selection. Refer to the section Selecting Text in the Editor .
Join Lines	Ctrl+Shift+J	Choose this command to join lines or literals .
Fill Paragraph		Choose this command to create soft wraps in a paragraph.
Duplicate Lines	Ctrl+D	Choose this command to duplicate a line or fragment of text. Refer to Adding, Deleting and Moving Code Elements .
Indent Selection/Unindent Selection	Tab / Shift+Tab	Choose this command to change indentation of the line at caret. Refer to the section Changing Indentation .
Toggle Case	Ctrl+Shift+U	Choose this command to change case of the selection. See Toggling Case .
Convert Indents		Point to this node to reveal the sub-menu of the possible indentation and toggle indentation style. Refer to Changing Indentation .
Encode XML/HTML Special Characters		Choose this command to convert the selected special character to its HTML name in the format `&char;` .
Edit as Table		Choose this command to invoke the table editor for the current documents.

View

Menu item	Keyboard shortcut	Description
Tool Windows		Point to this node to reveal the list of the available tool windows. Refer to the section Manipulating the Tool Windows .
Quick Definition	Ctrl+Shift+I	Choose this command to open the quick definition pop-up. Refer to the section Viewing Definition .
Quick Documentation	Ctrl+Q	Choose this command to view quick documentation pop-up window.
Parameter Info	Ctrl+P	Choose this command to view method parameter information .
Context Info	Alt+Q	Choose this command to show the current cursor position , if it runs out of the visible editor pane.
Recent Files	Ctrl+E	Choose this command to show the pop-up list of recently opened files and tool windows, and navigate to them.
Recently Changed Files	Ctrl+Shift+E	Choose this command to show the pop-up list of recently changed files and navigate to them .
Recent Changes	Shift+Alt+C	Choose this command to open the pop-up list of recent changes .
Compare with Clipboard		Choose this command to compare the file currently opened in the editor with the contents of the system clipboard. See Comparing Files .
Quick Switch Scheme	Ctrl+Back Quote	Choose this command to switch between schemes .
Toolbar		Select or clear this command to show or hide the main toolbar .
Tool Buttons		Select or clear this command to show or hide the tool window buttons .
Status Bar		Select or clear this command to show or hide the Status toolbar .
Navigation Bar		Check this command to show or hide the Navigation bar .
Enter/Exit Presentation Mode		Choose this command to enter or exit presentation mode .
Enter/Exit Distraction Free Mode		Choose this command to enter or exit distraction-free mode .
Enter/Exit Full Screen		Choose this command to enter or exit full screen mode .

Navigate

Menu item	Keyboard shortcut	Description
Class/File/Symbol	Ctrl+N / Ctrl+Shift+N / Ctrl+Shift+Alt+N	Choose these commands to find and jump to a class, file, or symbol by name .
Custom Folding...	Ctrl+Alt+Period	Choose this command to navigate between custom regions .

Line...	Ctrl+G	Choose this command to navigate to the specified line of code .
Back/Forward	Ctrl+Alt+Left / Ctrl+Alt+Right	Choose these commands to go through the history of the recently navigated items . These commands are duplicated with and buttons on the main toolbar.
Last/Next Edit Location	Ctrl+Shift+Backspace	Choose these commands to jump to the latest edit location and back.
Bookmarks		Point to this node to reveal the sub-menu of commands related to using bookmarks .
Select In...	Alt+F1	Choose this command to select the desired component from the pop-up list of possible targets .
Jump to Navigation Bar	Alt+Home	Choose this command to navigate across your project using the Navigation Bar .
Declaration	Ctrl+B	Choose this command to jump to a declaration of a symbol .
Implementation(s)	Ctrl+Alt+B	Choose this command to jump to an implementation of a method .
Type Declaration	Ctrl+Shift+B	Choose this command to jump to the type declaration of a symbol .
Super Method	Ctrl+U	Choose this command to jump to a super method of the method at caret.
Test	Ctrl+Shift+T	Choose this command to navigate to an existing test, or create a test. See section Creating Tests .
Related Symbol...	Ctrl+Alt+Home	.
File Structure	Ctrl+F12	Choose this command to navigate through the source code using the File Structure view .
File Path	Ctrl+Alt+F12	See Navigating to File Path .
Type/Method/Call Hierarchy	Ctrl+H / Ctrl+Shift+H / Ctrl+Alt+H	Choose these commands to navigate using the hierarchy views. Refer to the sections Viewing Structure and Hierarchy of the Source Code .
Next/Previous Highlighted Error	F2 Shift+F2	Choose these commands to navigate through the highlighted errors .
Next/Previous Emmet Edit Point	Shift+Alt+Close Bracket / Shift+Alt+Open Bracket	Refer to the section Emmet Support for details.
Next/Previous Change	Ctrl+Shift+Alt+Down Ctrl+Shift+Alt+Up	Choose these commands to navigate though the change markers (when VCS integration is enabled).
Next/Previous Method	Alt+Down Alt+Up	Choose these commands to go up and down through the methods and tags .

Code

Menu item	Keyboard shortcut	Description
Override Methods...	Ctrl+O	Choose this command to override a method. See Overriding Methods of a Superclass .
Implement Methods...	Ctrl+I	Choose this command to implement a method. See Implementing Methods of an Interface .
Generate...	Alt+Insert	Choose this command to create a new element. See (depending on the context) Populating Projects , Generating Code .
Surround With...	Ctrl+Alt+T	Choose this command to surround a logical fragment with code construct .
Unwrap/Remove...	Ctrl+Shift+Delete	Choose this command to unwrap an expression from enclosing statements .
Completion		Point to this node to reveal the nested auto-completion commands.
Folding		Point to this node to reveal the nested folding commands.
Insert Live Template...	Ctrl+J	Choose this command to create code constructs by live templates .
Surround with Live Template...	Ctrl+Alt+J	Choose this command to create code constructs using surround templates .
Comment with Line Comment	Ctrl+Slash	Choose this command to comment an entire line of code. See Commenting and Uncommenting Blocks of Code .
Comment with Block Comment	Ctrl+Shift+Slash	Choose this command to comment out a block of code. See Commenting and Uncommenting Blocks of Code .
Reformat Code...	Ctrl+Alt+L	Choose this command to perform code reformatting. See Reformatting Source Code .
Auto-Indent Lines	Ctrl+Alt+I	Choose this command to change indentation .
Optimize Imports...	Ctrl+Alt+O	Choose this command to optimize import statements. See Optimizing Imports .
Rearrange Code		Choose this command to rearrange code according to the arrangement rules. See Rearranging Code Using Arrangement Rules .
Move Statement Up/Down	Ctrl+Shift+Up / Ctrl+Shift+Down	Choose this command to move a statement up or down .
Move Parameter Left/Right	N/A / N/A	Choose this command to move an element in the list of parameters left or right .
Move Statement Down/Up	Ctrl+Shift+Down / Ctrl+Shift+Up	Choose this command to move a statement at caret down or up.
Move Line Up/Down	Shift+Alt+Up / Shift+Alt+Down	Choose this command to move a line at caret up or down .

Analyze

Menu item	Keyboard shortcut	Description
Inspect Code...		Choose this command to run an inspection .
Code Cleanup		Choose this command to open the dialog Specify Code Cleanup Scope Dialog .

Run Inspection by Name...	Ctrl+Shift+Alt+I	Choose this command to run the specified inspection .
Configure Current File Analysis...	Ctrl+Shift+Alt+H	Choose this command to change highlighting level of the current file .
View Offline Inspection Results...		Choose this command to see inspection results stored on your computer. See Viewing Offline Inspections Results .
Locate Duplicates...		Choose this command to find code duplicates. Refer to Analyzing Duplicates .
Analyze Stacktrace...		Choose this command to analyze external stacktrace .

Refactor

Note that the composition of this menu item depends upon the current context.

Menu item	Keyboard shortcut	Description
Refactor This...	Ctrl+Shift+Alt+T	Choose this command to open a pop-up menu of the refactorings available in the current context. Refer to the section Refactoring Source Code .
Rename...	Shift+F6	Choose this command to rename an element .
Change Signature...	Ctrl+F6	Choose this command to perform change signature refactoring .
Move...	F6	Choose this command to move a symbol to the specified location.
Copy...	Ctrl+C	Choose this command to create a copy of an element in the specified location.
Safe Delete	Alt+Delete	Choose this command to delete a symbol , performing search for its usages.
Extract		Choose this command to perform one of the extract refactorings. See Extract Refactorings for details.
Inline...	Ctrl+Alt+N	Choose this command to perform inline refactoring .
Pull Members Up...		Choose this command to perform pull members up refactoring .
Push Members Down		Choose this command to perform push members down refactoring .
Invert Boolean		Choose this command to perform invert boolean refactoring .
XML Refactorings		Point to this node to reveal the sub-menu of XML-related refactorings, available in the curremt context.

Build

Refer to the section Compiler and Builder for details.

Menu item	Keyboard shortcut	Description
Make Project	Ctrl+F9	Choose this command to make project .
Make Module <name>		Choose this command to make module .
Recompile <name>	Ctrl+Shift+F9	Choose this command to compile the specified target .
Rebuild Project		Choose this command to rebuild your project .
Generate Ant Build		Choose this command to generate an Ant build file .
Build Artifacts		Choose this command to Package a module into a JAR file .
Generate Signed APK		Choose this command to Generate a signed Android application package .
Run Ant Target	Ctrl+Shift+F10	Choose this command to execute an Ant target .

Run

Menu item	Keyboard shortcut	Description
Run <current run/debug configuration>	Shift+F10	Choose this command to run the current class with the `main()` method with the corresponding temporary run/debug configuration. This command is duplicated with icon on the main toolbar.
Debug <current run/debug configuration>	Shift+F9	Choose this command to debug the current class with the `main()` method with the corresponding temporary run/debug configuration. This command is duplicated with icon on the main toolbar.
Run <current run/debug configuration> with Coverage		Choose this command to run with coverage the current class with the `main()` method with the corresponding temporary run/debug configuration. This command is duplicated with icon on the main toolbar.
Run...	Shift+Alt+F10	Choose this command to select the desired run/debug configuration, and then launch it. Refer to the section Creating and Editing Run/Debug Configurations .
Debug...	Shift+Alt+F9	Choose this command to select the desired run/debug configuration, and then launch it in debugging mode. Refer to the section Creating and Editing Run/Debug Configurations .
Edit Configurations...		Choose this command to change run/debug configurations .
Import Test Results		Choose this command to import test results from a file.
Stop	Ctrl+F2	Choose this command to terminate execution of a run/debug configuration . This command is duplicated with icon in the toolboxes of the Run and Debug tool windows.
Show Running List		Choose this command to display a popup that lists all currently running applications. This option is disabled if there are no active applications at the moment.
Stepping Commands		These commands become enabled with the debugger session on. Refer to the section Stepping Through the Program . See

		also descriptions of the stepping toolbar buttons in the Debug tool window reference.
Pause Program		Choose this command to pause output of the current run or debug session. This command is duplicated with icon in the toolboxes of the Run and Debug tool windows.
Resume Program	F9	Choose this command to resume the debugger session with the selected run/debug configuration. This command is duplicated with icon in the toolbox of the Debug tool windows.
Evaluate Expression	Alt+F8	Choose this command to evaluate expression during the debug session.
Quick Evaluate Expression	Ctrl+Alt+F8	Choose this command to perform quick evaluation of an expression in the editor during the debug session.
Show Execution Point	Alt+F10	Choose this command to show execution point during the debug session.
Toggle Line Breakpoint	Ctrl+F8	Choose this command to turn on or off a line breakpoint. Refer to the section Creating Line Breakpoints .
Toggle Temporary Line Breakpoint	Ctrl+Shift+Alt+F8	Choose this command to turn on or off a temporary line breakpoint. Refer to the section Creating Line Breakpoints .
View Breakpoints...	Ctrl+Shift+F8	Choose this command to show all available breakpoints and change them in the Breakpoints dialog .

Tools

Note that composition of the menu Tools depends on the enabled plugins and external tools .

Menu item	Keyboard shortcut	Description
Tasks and Contexts		Point to this node to reveal the sub-menu of commands related to tasks and contexts management . See also Tasks reference page.
Save Project as Template		The the current project or any its module is saved as a template project .
Save File as Template		Choose this command to save the current file as a template file .
New Scratch File	Ctrl+Shift+Alt+Insert	Choose this command to create a new scratch file .
IDE Scripting Console		Choose this command to launch the interactive scripting console.
XML Actions		Point to this node to reveal the sub-menu of XML-related commands .
Capture Memory Snapshot		Choose this command to get the memory state of the profiled application.
Vim Emulator		Select this check command to enable or disable Vim emulation. This command only appears when Vim plugin is installed and enabled. Refer to the tutorial Configuring IntelliJ IDEA to work as a Vim editor .
Reconfigure Vim Keymap		This command is only visible, when Vim Emulator is checked. Choose it to select a different base keymap for the Vim emulator.
Deployment		Point to this node to reveal the sub-menu of deployment-related commands. Refer to the section Deployment. Working with Web Servers .
Open terminal		Choose this command to run the embedded local terminal .
Start SSH Session		Choose this command to launch a terminal on a remote SSH server. Refer to the section Running SSH Terminal .
Test RESTful Web Service		Choose this command to compose and run requests to a RESTful web service. Refer to the section Testing RESTful Web Services .
Kotlin		Point to this node to reveal the sub-menu of Kotlin-related commands . See also the Kotlin page https://kotlinlang.org/ .

VCS

Note that the VCS menu contains different commands, depending on the enabled version control system. The following table shows the menu commands available when no version control integration is enabled.

Menu item	Keyboard shortcut	Description
Local History		Point to this node to reveal the list of commands related to working with the Local History .
Enable Version Control Integration...		Choose this command to associate a project root with one of the supported version control systems.
VCS Operations Popup	Alt+Back Quote	Choose this command to invoke the pop-up list of the most popular VCS actions .
Apply Patch...		Choose this command to apply a patch .
Checkout from Version Control		Point to this node to reveal the sub-menu of the checkout commands, specific for the supported version control systems. With no version control integration enabled, it is possible to check out from SVN , Mercurial , Git , GitHub , and CVS .
Import into Version Control		Point to this node to reveal the sub-menu of the import commands, specific for the supported version control systems. With no version control integration enabled, it is possible to import to SVN , Mercurial , Git , GitHub , and CVS .
Browse VCS Repository		Point to this node to reveal the sub-menu of the browse commands, specific for the supported version control systems. With no version control integration enabled, it is possible to browse Subversion, Git, and CVS repositories that are not associated with the currently opened project. – Browse CVS Repository: when you choose this option IntelliJ IDEA opens the Select CVS Root Configuration dialog, where you can select the relevant CVS root, see Configuring CVS Roots and Browsing CVS Repository . – Browse Git Repository Log: choose this option to view the log for a local Git repository that is associated with another project. When you select the relevant repository in the Select Path dialog box, IntelliJ IDEA adds a new Log tab to the Version Control tool window and shows the log for the selected repository. The name of the project associated with the selected repository is displayed in the tab title, when you hover the mouse over this tab, the full path to the repository is shown in a tooltip. – Browse Subversion Repository: when you choose this option IntelliJ IDEA opens the SVN Repositories tool window where you can view, add, and edit location of SVN repositories, see Browsing Subversion Repository and Browsing Contents of the Repository .

It is important to note that with VCS integration enabled, the composition of the VCS menu is different. Refer to the following help sections for details:

Version Control concepts

- Version Control concepts
- Common VCS procedures
- VCS-specific procedures
- Version Control reference

Window

Menu item	Keyboard shortcut	Description
Store Current Layout as Default		Choose this command to save the current way the tool windows are arranged .
Restore Default Layout	Shift+F12	Choose this command to restore the initial way the tool windows are arranged .
Active Tool Window		Choose this command to reveal the sub-menu of commands, related to the active tool window . Refer to the sections Tool Windows , Manipulating the Tool Windows , Specifying the Appearance Settings for Tool Windows , Viewing Modes .
Editor Tabs		Choose this command to reveal the sub-menu of commands, related to the editor tabs. Refer to the section Managing Editor Tabs . Note that these commands can be also found on the context menu of an editor tab.
Background Tasks		Choose this command to reveal the sub-menu of commands, related to performing tasks in background. Refer to the section Working with Background Tasks .
Next Project Window / Previous Project Window	Ctrl+Alt+Close Bracket Ctrl+Alt+Open Bracket	Choose this command to switch between currently opened projects. See Switching Between Projects .
<project>		Select project to be shown in the active window. See Switching Between Projects .

Help

Menu item	Keyboard shortcut	Description
Find Action	Ctrl+Shift+A	Choose this command to invoke an action by its name .
Demos and Screencasts		Choose this command to open the IntelliJ IDEA YouTube page.
Help Topics		Choose this command to show the help topics. Refer to the section Using Help Topics .
Tip of the Day		Choose this command to show an arbitrary tip. Refer to the section Using Tips of the Day .
Productivity Guide		Choose this command to show productivity guide .
Support Center		Choose this command to open JetBrains Support page.
Submit Feedback		Choose this command to report your overall impression of IntelliJ IDEA to the support service. Refer to the section Reporting Issues and Sharing Your Feedback .
Show Log in Explorer/Finder		Choose this command to find IntelliJ IDEA's log. Refer to the section Reporting Issues and Sharing Your Feedback for details.
Edit Custom Properties		Choose this command to open the custom file `idea.properties` , located under the user home. If this file does not exist, IntelliJ IDEA suggests to create it. Refer to the section Tuning IntelliJ IDEA for details.
Edit Custom VM Options		Choose this command to open the custom file `*.vmoptions` , located under the user home. If this file does not exist, IntelliJ IDEA suggests to create it. Refer to the section Tuning IntelliJ IDEA for details.
Debug Log Settings		Choose this command to change logging level for a category. Choosing this command leads to opening the Custom Debug Log Configuration dialog box, where you have to type the log categories names, separated with new lines. Refer to the section Reporting Issues and Sharing Your Feedback .
Keymap Reference		Choose this command to open the default keymap in PDF format. Refer to the section Keymap Reference for details.
What's New in IntelliJ IDEA		Choose this command to open the What's New page .
Show Licences		Choose this command to show the legal information.
Diagnostics		This command and the items below it are available only when the Performance plugin is installed and activated. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins . Use the commands under this menu item to automatically capture snapshots and collect other data related to performance and send this data to JetBrains Support . When you click Diagnostics, a drop-down menu with the following items is displayed: – Enable CPU Profiler: choose this option to activate the CPU profiler and capture CPU snapshots. The CPU profiles remains active and the option not available till you restart IntelliJ IDEA. – Change Memory Settings: choose this option to edit the `-Xmx` setting and thus increase the heap size (the amount of available memory), see Tuning IntelliJ IDEA for details. – Execute Performance Script: choose this option to run special predefined scenarios with the CPU profiler enabled. After a scenario is executed it generates a CPU snapshot that can be used by the support team to analyze your performance issue. – Profile Indexing: when you choose this option, CPU profiler is active and captures a CPU snapshot from the moment you reopen the current project till the moment indexing is completed. The snapshot can be used by the support team to analyze a performance issue that arize during indexing. – Show Performance Summary: choose this option to collect most relevant information that can affect performance and at the same time does not reveal any private info about the current project and the current file. Learn more at Performance testing plugin in PhpStorm

Settings Summary	Choose this command to collect the IntelliJ IDEA settings and show all sorted by category so it is easy to locate the settings that are related with a specific problem (debugger, or deployment, or something else does not work) and send this summary to JetBrains Support . To open a request form, click Open Support Request in the dialog box that opens. This command is available only when the Performance plugin is installed and activated. The plugin is not bundled with IntelliJ IDEA, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins .
New Support Request	Choose this command to open the support ticket page. Fill out the fields (in particular, use the Settings Summary copy), and click Submit.
JetBrains TV	Choose this command to open JetBrains TV page .
Register	Choose this command to register IntelliJ IDEA.
Check for Update	Choose this command to obtain information about the current version, and the availability of newer versions of IntelliJ IDEA. Refer to Updates page. This command in available on Windows/Linux. On Mac OS it appears on the IntelliJ IDEA menu.
About	Choose this command to obtain information about the current version of IntelliJ IDEA, current build, etc. Press Escape to close the pop-up window. This command in available on Windows/Linux. On Mac OS it appears on the IntelliJ IDEA menu.

Tuning IntelliJ IDEA

In this part:

- Tuning IntelliJ IDEA
  - Changing IntelliJ IDEA properties
  - Managing *.vmoptions file
    - Example: Increasing the heap size
  - Managing idea.properties file
    - Example: Changing case of unicode literals
  - Specifying custom JDK, properties or vmoptions files across platforms
- File IDEA.Properties'
- Project and IDE Settings
- Directories Used by IntelliJ IDEA to Store Settings, Caches, Plugins and Logs
- Synchronizing and Sharing Settings
- Networking in IntelliJ IDEA
- Color-Deficiency Adjustment

Changing IntelliJ IDEA properties

IntelliJ IDEA makes it possible to change `*.vmoptions` and `idea.properties` files without editing them in the IntelliJ IDEA installation folder.

To create an empty `idea.properties` file or to copy `*.vmoptions` file, choose the Help | Edit Custom Properties or Help | Edit Custom VM Options on the main menu respectively. Refer to the menu items description for details.

Managing *.vmoptions file

The location of the `*.vmoptions` file depends on the operating system you are currently using:

- For Windows systems: `<IntelliJ IDEA installation folder>/bin/idea.vmoptions` or `<IntelliJ IDEA installation folder>/bin/idea64.vmoptions`
- For *NIX systems: `<IntelliJ IDEA installation folder>/bin/idea.vmoptions` or `<IntelliJ IDEA installation folder>/bin/idea64.vmoptions`
- For OS X systems, you have to make a copy of the `idea.vmoptions` file in the IDE preferences folder and then edit this copy. The reason is that app bundle is signed and you should not modify any files inside the bundle.

  Since version 12.0.0:

  The file `/Applications/IntelliJ Idea XX.app/bin/idea.vmoptions` or `/Applications/IntelliJ Idea CE XX.app/bin/idea.vmoptions` should be copied to `~/Library/Preferences/IntelliJIdeaXX/idea.vmoptions` or `~/Library/Preferences/IdeaICXX/idea.vmoptions`

  Since version 14.0.0:

  The file `/Applications/IntelliJ Idea XX.app/Contents/bin/idea.vmoptions` or `/Applications/IntelliJ Idea CE XX.app/Contents/bin/idea.vmoptions` should be copied to `~/Library/Preferences/IntelliJIdeaXX/idea.vmoptions` or `~/Library/Preferences/IdeaICXX/idea.vmoptions`.

  For the older versions, the settings are stored in `/Applications/IntelliJ IDEA <version>.app/Contents/Info.plist`.

Tip Mind the abbreviation scheme of the IntelliJ IDEA name: `XX` corresponds to the IntelliJ IDEA version.

To avoid editing files in the IntelliJ IDEA installation folder, one should:

1. Do one of the following:
   - Use the main menu command Help | Edit Custom VM Options to create a copy of the `idea.vmoptions` file in the user home. Refer to Edit Custom VM Options for details.
   - Copy the existing file from the IntelliJ IDEA installation folder somewhere and save the path to this location in the environment variable `IDEA_VM_OPTIONS`.
   - Copy the existing file `<IntelliJ IDEA installation folder>/bin/idea$BITS.vmoptions` from the IntelliJ IDEA installation folder into the location under user home.
2. Edit this file in the new location.

If `IDEA_VM_OPTIONS` environment variable is defined, or the `*.vmoptions` file exists, then this file is used instead of the one located in the IntelliJ IDEA installation folder.

Example: Increasing the heap size

To increase IntelliJ IDEA heap size, you should copy the original `idea.vmoptions` file to the above-mentioned location, and then modify the `-Xmx` setting.

Managing IDEA.properties file

The file `idea.properties`, located in the `bin` directory of the IntelliJ IDEA installation folder, should not be edited. Instead of editing the original `idea.properties`, create file `idea.properties` in the location specified below, open it for editing and add the required properties.

So, depending on your platform:

- For Windows: in `%USERPROFILE%\.IntelliJIdeaXX` or `%USERPROFILE%\.IdeaICXX`
- For *NIX: in `~/.IntelliJIdeaXX` or `~/.IdeaICXX`
- For OS X: in `~/Library/Preferences/IntelliJIdeaXX` or `~/Library/Preferences/IdeaICXX`

Example: Changing case of unicode literals

IntelliJ IDEA allows defining whether non-ascii characters should use literals like `'\u00AB'` or `'\00ab'`.

This behavior is controlled by the system property `idea.native2ascii.lowercase`. By default, upper case characters are used.

If it is desirable to use lower case characters, create the file `idea.properties` in the location specified above, open it for editing and add the following line:

```
idea.native2ascii.lowercase=true
```

Specifying custom JDK, properties or vmoptions files across platforms

A custom JDK, `*.properties` and `*.vmoptions` files are specified across platforms in a unified way.

All the launchers look at the following environment variables:

- `$<IDE-NAME>_JDK` (`<IDE-NAME>_JDK_64`)
- `$<IDE-NAME>_PROPERTIES`
- `$<IDE-NAME>_VM_OPTIONS`

File 'idea.properties'

The file `idea.properties`, located in the `bin` directory of the IntelliJ IDEA installation folder, should not be edited. Instead of editing the original `idea.properties`, create file `idea.properties` in the location specified below, open it for editing and add the required properties.

So, depending on your platform:

- For Windows: in `%USERPROFILE%\.IntelliJIdeaXX` or `%USERPROFILE%\.IdeaICXX`
- For *NIX: in `~/.IntelliJIdeaXX` or `~/.IdeaICXX`
- For OS X: in `~/Library/Preferences/IntelliJIdeaXX` or `~/Library/Preferences/IdeaICXX`

You can create an empty file `idea.properties` and open it in the editor by choosing the Help | Edit Custom Properties command on the main menu.

Refer to the description of Edit custom properties command.

Name	Description	Property setting
Note	The Windows users should use forward slashes. i.e. `c:/idea/system`.	
${idea.home} macro	Use ${idea.home} macro to specify location relative to IDE installation home. Also use `${xxx}` where `xxx` is any java property (including defined in the previous lines of this file) to refer to its value.	
Path to the IDE config folder	Uncomment this option if you want to customize path to IDE config folder.	`idea.config.path=${user.home}/.IntelliJ IDEA/config`
Path to IDE system folder	Uncomment this option if you want to customize path to IDE system folder.	`idea.system.path=${user.home}/.IntelliJ IDEA/system`
Path to user installed plugins	Uncomment this option if you want to customize path to user installed plugins folder.	`idea.plugins.path=${user.home}/.IntelliJ IDEA/config/plugins`
Path to IDE logs folder	Uncomment this option if you want to customize path to IDE logs folder.	`idea.log.path=${user.home}/.IntelliJ IDEA/system/log`
Maximum file size	Maximum file size (kilobytes) IDE should provide code assistance for. The larger file is, the slower its editor works and higher overall system memory requirements are, if code assistance is enabled. Remove this property or set to a very large number, if you need code assistance for any files to be available, regardless of their size.	`idea.max.intellisense.filesize=2500`
Console cyclic buffer	This option controls console cyclic buffer: keeps the console output size not higher than the specified buffer size (Kb). Older lines are deleted. In order to disable cycle buffer, use `idea.cycle.buffer.size=disabled`	`idea.cycle.buffer.size=1024`
Launcher	Configure if a special launcher should be used when running processes from within IDE. Using Launcher enables "soft exit" and "thread dump" features.	`idea.no.launcher=false`
Classpath	To avoid too long classpath	`idea.dynamic.classpath=false`
ProcessCanceledException	Uncomment this property to prevent IDE from throwing ProcessCanceledException when user activity detected. This option is only useful for plugin developers, while debugging PSI related activities performed in background error analysis thread. DO NOT UNCOMMENT THIS UNLESS YOU'RE DEBUGGING THE IDE ITSELF. Significant slowdowns and lockups will happen otherwise.	`idea.ProcessCanceledException=disabled`

Pop-up window weight	There are two possible values of idea.popup.weight property: "heavy" and "medium". If you have WM configured as "Focus follows mouse with Auto Raise" then you have to set this property to "medium". It prevents problems with the pop-up menus on some configurations.	`idea.popup.weight=heavy`
System anti-aliasing	Use default anti-aliasing in system, i.e. override value of "Settings|Editor|Appearance|Use anti-aliased font" option. May be useful when using Windows Remote Desktop Connection for instance.	`idea.use.default.antialiasing.in.editor=false`
Repaint	Disabling this property may lead to visual glitches like blinking and fail to repaint on certain display adapter cards.	`sun.java2d.noddraw=true`
Editor performance	Removing this property may lead to editor performance degradation under Windows.	`sun.java2d.d3d=false`
Slow scrolling	Workaround for slow scrolling in JDK6.	`swing.bufferPerWindow=false`
Editor performance under X Window	Removing this property may lead to editor performance degradation under X Window.	`sun.java2d.pmoffscreen=false`
Avoid long hangs	Workaround to avoid long hangs while accessing clipboard under OS X.	`ide.mac.useNativeClipboard=True`
Maximum load size	Maximum size (kilobytes) IDEA will load for showing past file contents - in Show Diff or when calculating Digest Diff	`idea.max.vcs.loaded.size.kb=20480`
Copy library jars	IDE copies library jars to prevent their locking. If copying is not desirable, specify "true"	`idea.jars.nocopy=false`
Start the JVM in debug mode	The VM option value to be used to start the JVM in debug mode. Some JREs define it in a different way (-XXdebug in Oracle VM)	`idea.xdebug.key=-Xdebug`
Switch into JMX 1.0 compatibility mode.	Uncomment this option to be able to run IntelliJ IDEA using J2SDK 1.5+ while working with application servers (like WebLogic) running 1.4.	`jmx.serial.form=1.0`
Fatal errors notifications	Change to 'enabled' if you want to receive instant visual notifications about fatal errors that happen to an IDE or plugins installed.	`idea.fatal.error.notification=disabled`

Project and IDE Settings

In this topic:

– Overview
– Project Settings
– IDE Settings
– Locations of directories

Overview

There are two types of settings that define your preferred environment:

– Project Settings, that apply to a specific project. They are marked with in the Settings/Preferences dialog.
– IDE Settings, that are common for all projects and refer to the project-independent aspects.

Project Settings

Project settings are stored with each specific project as a set of `xml` files under the `.idea` folder. If you specify the default project settings , these settings will be automatically used for each newly created project.

The settings that pertain to a project, are marked with the icon in the Settings/Preferences dialog.

IDE Settings

IDE settings are stored in the dedicated directories under IntelliJ IDEA home directory. The IntelliJ IDEA directory name is composed of the product name and version.

For IntelliJ IDEA Community edition the folder name is `.IdeaICXX` .

For example:

Windows

– `<User home>\.IntelliJIdeaXX\config` that contains user-specific settings.
– `<User home>\.IntelliJIdeaXX\system` that stores IntelliJ IDEA data caches.

`<User home>` in WindowsXP is `C:\Documents and Settings\<User name>\` ; in Windows Vista it is `C:\Users\<User name>\`

Linux

– `/.IntelliJIdeaXX/config` that contains user-specific settings.
– `~/.IntelliJIdeaXX/system` that stores IntelliJ IDEA data caches.

OS X

– `~/Library/Application Support/IntelliJIdeaXX` contains the catalog with plugins.
– `~/Library/Preferences/IntelliJIdeaXX` contains the rest of the configuration settings.
– `~/Library/Caches/IntelliJIdeaXX` contains data caches, logs, local history, etc. These files can be quite significant in size.
– `~/Library/Logs/IntelliJIdeaXX` contains logs.

Locations of directories

The `config` directory has several subfolders that contain xml files with your personal settings. You can easily share your preferred keymaps, color schemes, etc. by copying these files into the corresponding folders on another IntelliJ IDEA installation. Prior to copying, make sure that IntelliJ IDEA is not running, because it can erase the newly transferred files before shutting down.

The following is the list of some of the subfolders under the `config` folder, and the settings contained therein.

Folder name	User Settings
`codestyles`	Contains code style schemes .
`colors`	Contains editor colors and fonts customization schemes.
`filetypes`	Contains user-defined file types .
`inspection`	Contains code inspection profiles .
`keymaps`	Contains IntelliJ IDEA keyboard shortcuts customizations.
`options`	Contains various options, for example, feature usage statistics and macros.
`templates`	Contains user-defined live templates .
`tools`	Contains configuration files for the user-defined external tools .
`shelf`	Contains shelved changes .

Locations of the `config` , `system` , `plugins` directories can be modified in `idea.properties` file.

IntelliJ IDEA makes it possible to change `*.vmoptions` and `idea.properties` files without editing them in the IntelliJ IDEA installation folder.

To create an empty `idea.properties` file or to copy `*.vmoptions` file, choose the Help | Edit Custom Properties or Help | Edit Custom VM Options on the main menu respectively. Refer to the menu items description for details.

To learn how to change the `idea.properties` file, read the section File 'idea.properties' .

Directories Used by IntelliJ IDEA to Store Settings, Caches, Plugins and Logs

Location of the IDE files depends on the operating system, and IntelliJ IDEA version.

In this section:

– Windows
– Linux and the other UNIX systems
– OS X

Windows

All the files are located under this directory by default:

Windows Vista, 7, 8, 10

```
<SYSTEM DRIVE>\Users\<USER ACCOUNT NAME>\.<PRODUCT><VERSION>
```

Windows XP

```
<SYSTEM DRIVE>\Documents and Settings\<USER ACCOUNT NAME>\.<PRODUCT><VERSION>
```

Example

– IntelliJ IDEA 2016.1 Ultimate Edition:

```
c:\Users\John\.IntelliJIdea2016.1\
```

– IntelliJ IDEA 2016.1 Community Edition

```
c:\Users\John\.IdeaIC2016.1\
```

Refer to the page Project and IDE Settings .

Linux and the other UNIX systems

Product directory starting with dot can be found in the user home directory. The pattern is:

```
~/.<PRODUCT><VERSION>
```

~ is an alias for the home directory, for example `/home/john` .

OS X

– Configuration

```
~/Library/Preferences/<PRODUCT><VERSION>
```

– Caches

```
~/Library/Caches/<PRODUCT><VERSION>
```

– Plugins

```
~/Library/Application Support/<PRODUCT><VERSION>
```

– Logs

```
~/Library/Logs/<PRODUCT><VERSION>
```

where <PRODUCT> is IntelliJIDEA (IntelliJ IDEA Ultimate Edition), IdeaIC (IntelliJ IDEA Community Edition).

Synchronizing and Sharing Settings

In this section:

- Synchronizing settings
- Prerequisite
- Basics
- What is synched?
- Connecting to IntelliJ Configuration Server
- Configuring login to IntelliJ Configuration Server
- Sharing settings

Synchronizing settings

Prerequisite

Before you start working with IntelliJ Configuration Server, make sure that the IntelliJ Configuration Server plugin is downloaded and enabled, as described in Installing, Updating and Uninstalling Plugins .

Basics

IntelliJ IDEA makes it possible to synchronize settings across several different IntelliJ IDEA installations, or auto-configure new installations, using settings from the server.

Once logged in, the settings are synchronized with the server based on "updated" timestamp stored in each settings group. The most recent ones are used and are either sent to server or applied locally from the server.

IntelliJ IDEA always works with settings stored locally. So, the local settings are used, if, at some moment, the server is not available. On the following connection all updated settings will be pushed to the server.

When logged in to the server, once some settings are updated locally inside IntelliJ IDEA, they are subsequently sent to the server. But local settings are only updated on IntelliJ IDEA startup.

The server stores only one set of settings for each user.

What is synched?

IntelliJ Configuration Server stores almost all of the IDE and project settings, except for some platform-specific ones, and those containing local paths. This includes code style settings, keymaps, fonts and colors scheme, inspection profiles, and others.

Connecting to IntelliJ Configuration Server

You can connect to the IntelliJ Configuration Server in various ways:

- During the first IntelliJ IDEA startup after installing the plugin, through the Login to IntelliJ Configuration Server dialog box.
- At any time during a session through the Login to IntelliJ Configuration Server dialog box.
- During IntelliJ IDEA startup according to the previously configured settings :
  - Silently
  - Through the Login to IntelliJ Configuration Server dialog box

To connect to the IntelliJ Configuration Server during a session

1. At any time during a session, click the IntelliJ Configuration Server Status button (when connected to the server, or , when disconnected) on the Status bar.
2. In the dialog box that opens, click the Login button.
3. Specify the login and password of your JetBrains Account or create a JetBrains Account if you do not have it.
   To connect to the IntelliJ Configuration Server through a Proxy server, select the Use HTTP Proxy check box and specify the Proxy server parameters in the dialog box controls. See a detailed description of controls in the Login to IntelliJ Configuration Server dialog box reference.

Configuring login to IntelliJ Configuration Server

To configure login to the Server during the next IntelliJ IDEA startup

1. Click the IntelliJ Configuration Server Status button / on the Status bar.
2. In the dialog box that opens, specify whether you want to log in to the IntelliJ Configuration Server during the next startup and how you want to do it. The following options are available:
   - Show login dialog box
   - Login silently
   - Do not login

   Select the desired option and click OK.

Tip The specified configuration settings will be applied during the next IntelliJ IDEA startup.

Sharing settings

For sharing project settings, `.idea` project configuration directory should be shared via version control .

If you decide to share IntelliJ IDEA project files with the other developers, follow these guidelines.

Directory based project format (`.idea` directory) is used by all the recent IntelliJ IDEA versions by default. Here is what you need to share:

– All the files under `.idea` directory in the project root, except the `workspace.xml` and `tasks.xml` files, which store user-specific settings.

– All the `.iml` module files that can be located in different module directories.

Be careful about sharing the following:

– Android artifacts that produce a signed build (will contain keystore passwords)

– `dataSources.ids` , `datasources.xml` (can contain database passwords)

You may consider not to share the following:

– `gradle.xml` .

– XML files under `.idea/libraries` in case they are generated from Gradle project.

– user `dictionaries` folder (to avoid conflicts if another developer has the same name).

For the legacy project format (.ipr/.iml/.iws files), share the project `.ipr` file and all the `.iml` module files, don't share the `.iws` file, since it stores user specific settings.

Networking in IntelliJ IDEA

IntelliJ IDEA requires Internet connection for a wide variety of tasks. For example:

- Checking for IntelliJ IDEA updates
- Code inspections that can verify external resources
- Communication with the version control servers, task servers
- Anonymous usage statistics
- Resolving Maven dependencies and updating plugins
- Using the IDEtalk plugin

Besides that, IntelliJ IDEA provides IPC for commands (for example, open files), and the built-in Web server.

Some of the communication requirements are configurable:

- Checking for updates can be turned off. To disable checking for updates, open the page Settings | Updates , and clear the check box Check for updates in channel.
- To turn off IDEtalk plugin, open the page Settings | Plugins , and make sure that the check box to the left of the plugin name is cleared.
- To use only those Maven resources that are available locally, open the page Settings| Maven , and select the check box Work offline
- To disable code inspection that highlights dead links, open the page Settings | Inspections , and clear the check box to the left of the HTML inspection Non-existent web resources.
- You can control the frequency of sending usage statistics, or even completely disable this function in the page Settings | Usage statistics .

Color-Deficiency Adjustment

In this section:

– Light editor schemes

– Darcula scheme

– Test runner adjustment

Light editor schemes

To people with red-green color deficiency, green, red and their hues might look the same. In the default color scheme, red is reserved for the errors and green for the strings:

```
@NonNls private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@NotNull final Project project,
                                    @NotNull ApplicationProfileManager applicationProfileManager,
                                    @NotNull DependencyValidationManager holder)
  myProject = project;
  myHolder = holder;
  myApplicationProfileManager = applicationProfileManager;
}
```

The simulation below shows how the same code fragment will look for a person with green color deficiency. Strings, annotations and unknown symbols are all the same color. The wavy error underline is lighter and less noticeable:

```
@NonNls private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@NotNull final Project project,
                                    @NotNull ApplicationProfileManager applicationProfileManager,
                                    @NotNull DependencyValidationManager holder)
  myProject = project;
  myHolder = holder;
  myApplicationProfileManager = applicationProfileManager;
}
```

The Adjust for color deficiency option changes colors so that they can be differentiated by a person with green or red color deficiency. Strings and annotations are shades of blue, and orange is reserved for the error states:

```
@NonNls private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@NotNull final Project project,
                                    @NotNull ApplicationProfileManager applicationProfileManager,
                                    @NotNull DependencyValidationManager holder)
  myProject = project;
  myHolder = holder;
  myApplicationProfileManager = applicationProfileManager;
}
```

Then, with the option Adjust for color deficiency turned on, and green color deficiency is selected, the simulation for the green color deficiency will look like the following:

```
@NonNls private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@NotNull final Project project,
                                    @NotNull ApplicationProfileManager applicationProfileManager,
                                    @NotNull DependencyValidationManager holder)
  myProject = project;
  myHolder = holder;
  myApplicationProfileManager = applicationProfileManager;
}
```

Highlight for read/write states of identifiers at caret are well distinguished for the persons without color deficiency:

```
Class callerClass = null;
for (final Object o : element.getChildren()) {
  Element listElement = (Element)o;
  if (ATTR_LIST.equals(listElement.getName())) {
    if (callerClass == null) {
      callerClass = ReflectionUtil.findCallerClass(2);
      assert callerClass != null;
    }
```

However, these states become indistinguishable for the persons with color deficiency:

```
Class callerClass = null;
for (final Object o : element.getChildren()) {
  Element listElement = (Element)o;
  if (ATTR_LIST.equals(listElement.getName())) {
    if (callerClass == null) {
      callerClass = ReflectionUtil.findCallerClass(2);
      assert callerClass != null;
    }
```

With the Adjust for color deficiency option, the difference between read and write states remains visible. In the example below you see the green color deficiency simulation:

```
Class callerClass = null;
for (final Object o : element.getChildren()) {
  Element listElement = (Element)o;
  if (ATTR_LIST.equals(listElement.getName())) {
    if (callerClass == null) {
      callerClass = ReflectionUtil.findCallerClass(2);
      assert callerClass != null;
    }
```

Folded text highlight in the default scheme is easily noticeable for the persons without color deficiency:

```
String ABSTRACTION_GROUP_NAME = InspectionsBundle.message("group.names.abstraction.issues");
String ASSIGNMENT_GROUP_NAME = "Assignment issues";
String BUGS_GROUP_NAME = "Probable bugs";
```

But this is a challenge for the persons with color deficiency - it is too light to notice:

```
String ABSTRACTION_GROUP_NAME = InspectionsBundle.message("group.names.abstraction.issues");
String ASSIGNMENT_GROUP_NAME = "Assignment issues";
String BUGS_GROUP_NAME = "Probable bugs";
```

With the Adjust for color deficiency option it becomes noticeable:

```
String ABSTRACTION_GROUP_NAME = InspectionsBundle.message("group.names.abstraction.issues");
String ASSIGNMENT_GROUP_NAME = "Assignment issues";
String BUGS_GROUP_NAME = "Probable bugs";
```

Darcula scheme

The non-adjusted Darcula text looks as follows:

```
@NonNls private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@NotNull final Project project,
                                    @NotNull ApplicationProfileManager applicationProfileManager,
                                    @NotNull DependencyValidationManager holder)
  myProject = project;
  myHolder = holder;
  myApplicationProfileManager = applicationProfileManager;
}
```

The difference between the various elements of text is not noticeable for the persons with color deficiency:


```
@NonNls private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@NotNull final Project project,
                                    @NotNull ApplicationProfileManager applicationProfileManager,
                                    @NotNull DependencyValidationManager holder)
  myProject = project;
  myHolder = holder;
  myApplicationProfileManager = applicationProfileManager;
}
```


Now turn on the option Adjust for color deficiency.

Compare the view of the editor for the persons without color deficiency:

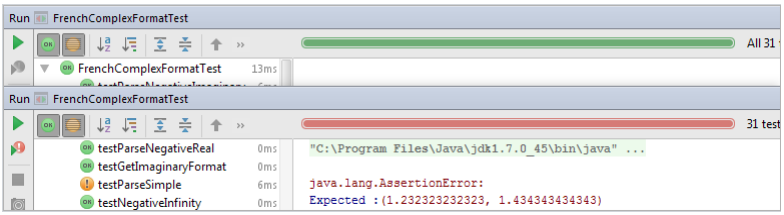

```
@NonNls private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@NotNull final Project project,
                                    @NotNull ApplicationProfileManager applicationProfileManager,
                                    @NotNull DependencyValidationManager holder)
  myProject = project;
  myHolder = holder;
  myApplicationProfileManager = applicationProfileManager;
}
```


and that for the persons with color deficiency:

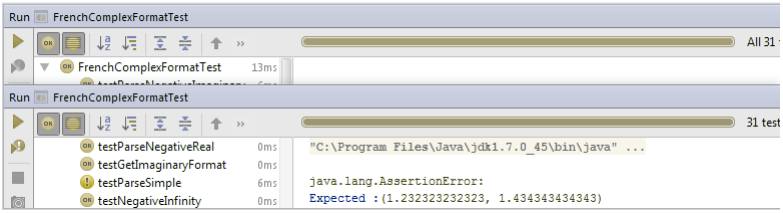

```
@NonNls private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@NotNull final Project project,
                                    @NotNull ApplicationProfileManager applicationProfileManager,
                                    @NotNull DependencyValidationManager holder)
  myProject = project;
  myHolder = holder;
  myApplicationProfileManager = applicationProfileManager;
}
```


Test runner adjustment

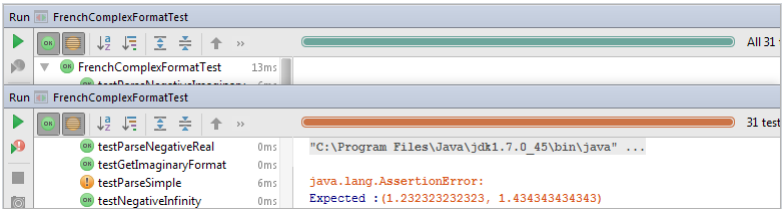
Colors of the test runner progress have also been adjusted. The usual progress bar color is indistinguishable for a person with green or red color deficiency. Compare the view of the test runner for the person without color deficiency:



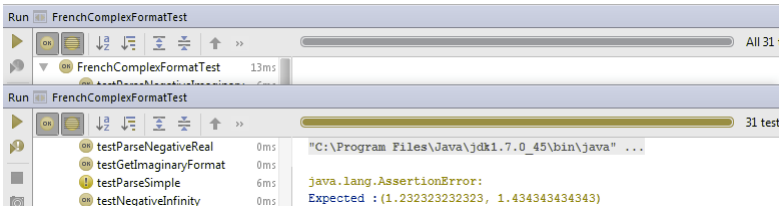
with that for the persons with color deficiency:



Turn on the option Adjust for color deficiency, and compare the view of the test runner for the person without color deficiency:



with that for a person with color deficiency:



Sending Feedback

Your feedback, including error reports, improvement suggestions, new feature requests and any other things you might have to say to JetBrains team, is welcome at the addresses listed below.

– JetBrains support

– Community support